Topology and Routing Aware Mapping on Parallel Processors

THESIS
submitted in partial fulfillment for the award of the degree of

Doctor of Philosophy
in
Computer Science

Devi Sudheer Kumar CH

DEPARTMENT OF MATHEMATICS & COMPUTER SCIENCE
Sri Sathya Sai Institute of Higher Learning
(Deemed to be University)
Prasanthi Nilayam, 515134, Ananthapur District, A.P., India

February 2013

Abstract

Communication costs of a typical parallel application increase with the number of processes. With the increasing sizes of current and future high-end parallel machines, communication costs can account for a significant fraction of the total run time even for massively parallel applications that are traditionally considered scalable. Hence, in order for applications to achieve the scalability for utilizing the very large number of processing nodes/cores in contemporary and future high end parallel machines, techniques must be developed to minimize the communication costs.

One of the important issues that need to be dealt with in the optimization of communication costs on large machines is the impact of topology and routing. Though small message latency is not very dependent on location in the machine, network contention can play a major role in limiting the bandwidth, even if the bisection bandwidth is high, due to the limitation of the routing scheme. Knowledge of topology and routing can be used to map the processes to nodes such that the inter-process communication lead to less contention on the network. The quality of a mapping is often measured using a metric called the hop-bytes metric, which gives an indication of the average communication load on each link in the network. We develop general mapping techniques by posing the hop-byte metric as a quadratic assignment problem (QAP). The QAP is a well studied NP-hard problem and the existing heuristics available for it can be used to solve the mapping problem. A limitation of the above metric is that in reality a link having the maximum load is often the bottleneck. Hence, our objective is to determine a mapping that minimizes the maximum load on any

given link. A metric based on this idea, called the maximum contention metric, requires the routing information along with the topology details. We showed that our heuristics for optimizing this metric is even more effective in reduction communication costs.

For scaling the applications on the large machines, addressing the data movement challenge within a node is almost as important as addressing it across the nodes as discussed above. The inter-core data movement overhead is strongly influenced by the assignment of threads to cores for many realistic communication patterns. We identified the bottlenecks to optimal performance by studying the interconnection network operational features and used this information to determine good affinities for standard communication patterns. Our study on the IBM Cell processor showed that the performance is up to a factor of two better than the default assignment. We also evaluated the affinity on a Cell blade consisting of two Cell processors, and developed a tool that can automatically select a suitable mapping for a given communication pattern.

Acknowledgements

I am truly grateful as I complete this thesis. I consider it a great pleasure to express my deep sense of love and gratitude to my most beloved Lord, Bhagawan Sri Sathya Sai Baba, Divine Founder Chancellor of this esteemed Institute, whose love, grace and blessings have inspired me to complete this work successfully. It is amazingly clear to me on how he has been the master divine director for this entire process of PhD.

This thesis would not have been possible without the support of many people. During the process of this research, I got opportunity to meet and come across the following people who made this thesis possible. It would not have been possible without the support of my parents and my dear sister to get me here so I would like first thank them for their support. Without their dedication and sacrifice, none of this would have been possible.

It has been truly an inspiring and enriching experience to be associated with Prof. Ashok Srinivasan, my research advisor. The role played by my him in shaping this thesis and my research work cannot be put in a few words. I admire him for the many great qualities that he is endowed with. He has kept a lot of quality time for me all these years, and doing all this remotely would have been tough for him. His expertise and breadth of knowledge provided great insights into research and made this thesis possible. I have been very fortunate to be able to have worked under his guidance.

I would like to thank each one of the current and the former members of the Department of Mathematics and Computer Science, for their help, encouragement and support throughout these years. The department has been an amazing place to work. My teachers at school, college and at SSSIHL have been a source of guidance and encouragement and I cannot thank them enough.

I express my sincere regards to the University and Campus Administration

for all the help and support they extended at every step of this research endeavor.

I would like to thank the technical staff at XSEDE for their excellent support in using the supercomputers. I must thank Dr. Radha Nandkumar for introducing me to the supercomputers. I would like to thank Prof. Arun Agarwal and Prof. Jaya Nair for serving as the chairs of my synopsis oral committee.

I would like to thank Mr. Shakti Kapoor, IBM, a visiting faculty to the Institute, for his enlightening lectures on Computer Architecture and Operating Systems, which have provided me a chance to build a sound technical base. I also thank Prof. Sadayappan, Ohio State University, for his resourceful lectures on Parallel Computing and also for his support while we worked together on a research work closely related to this thesis.

Many thanks to the students with whom I had the privilege of working with for some parts of this thesis. I would also like to acknowledge the students to whom I had the pleasure of teaching courses for being extremely supportive and understanding.

Many thanks also to the office staff who always worked hard in maintaining a pleasant environment in my workplace. I can not possibly name all the people who had positively contributed in making this endeavor of mine to reach where it is now. I convey my sincere thanks to all those people.

I finally wish to thank the reviewers for spending their valuable time in evaluating the thesis. Their suggestions and critical comments have enabled me to fix the gaps existed in the earlier version of the thesis. Due to their sincere efforts, the thesis now is in a better shape than it was earlier.

Credits

This thesis used allocations on the supercomputers at XSEDE centers. Without allocations on these machines, this work would not be as relevant and convincing.

I acknowledge the partial support by NSF grant # DMS-0626180. I thank IBM for providing access to a Cell Blade under the VLP program. I also thank Prof. David Bader for providing access to the Cell Blade at the Cell Center for Competence at Georgia Tech. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

Latex and gnuplot have been used in preparing this dissertation and are key to the writing of this document. Several other softwares and libraries were used in the process of working on my thesis and writing it. The script bargraph.pl developed by Derek Bruening has been used for creating nice bar graphs. I thank Dr. Abhinav Bhatele for providing inputs from the Mesh code, and also Prof. Torsten Hoefler for providing scripts to read the UFL sparse matrix collection inputs. Finally, the tool isosurface, part of the MATLAB, was used for visualizing the node allocations graphs.

Contents

List of Figures

List of Tables

1. Introduction

High performance computing power is believed to be the key to scientific & engineering leadership, industrial competitiveness, and national security. Major scientific discoveries and engineering breakthroughs are accelerated by utilizing world-leading computing facilities. Scientific applications range from drug discovery and genomics research, Oil exploration and energy research to weather forecasting and climate modeling. The escalating computational requirements of such applications has motivated the development of massively parallel machines, like the recent Blue Gene (BG/Q) machine from IBM and the Cray XE6 based Blue Waters machine at NCSA. Parallelism at the scale of millions of processors can be seen on these machines. The compute nodes in such machines are interconnected using tightly-coupled network. And hence it is essential that techniques for efficient and uniform utilization of the network resources be developed. Suitable mapping of tasks in a parallel application to nodes of these machines can substantially improve communication performance by reducing network congestion.

1.1 Motivation

The annual rate at which the time for flop is improving (59%) is far greater than the rate at which network link bandwidth is improving (26%). Hence, computation is cheap compared to communication and the network links are oversubscribed. The motivation for this research are the studies that are showing that communication costs can already exceed arithmetic costs by orders of magnitude, and the gap is growing exponentially over time. As massively parallel computers become larger, the interconnect topology will play a more significant role in determining the communication performance. Although, ideally, we would not want to use the topology information since this will result in techniques that are specific to a topology or an architecture. However, at the scale of millions of processing cores, the impact of topology and routing is so significant that they must be taken into consideration in order to achieve necessary scalability.

Some supercomputers such as Cray's XT5 [1], XK6 [2] and IBM Blue Gene/P [3] machines have 3D torus topologies. In such computers, minimizing network contention by matching communication pattern with the topology is critical for the communication performance [4]. Other current massively parallel computers such as the TACC Ranger use the nonblocking fat-tree topology. Although the fat-tree is nonblocking, the network has contention especially with static routing in InfiniBand networks [5]. Hence, for massively parallel computers, network contention can have a significant impact on performance: to minimize network contention, topology and routing must be taken into consideration.

There are several ways the topology and routing can be used for optimizing communication performance. In this work, we consider topology and routing aware process assignment. Depending on application communication pattern, the mapping from logical MPI processes to physical cores determines the physical communication. Recent works [4, 9, 10, 11, 12, 13, 14, 15, 16] have shown substantial communication performance improvement on large parallel machines by suitable assignment of processes or tasks to nodes of the machine. Earlier works on graph embedding are usually not suitable for modern machines because the earlier works used metrics suitable for a store-and-forward communication mechanism.

On modern machines on the other hand, in the absence of network congestion, latency is quite independent of location; communication performance is limited by contention on specific links. Yet another significant difference is that the earlier works typically embedded graphs onto standard network topologies such as hypercubes and meshes. On massively parallel machines, jobs typically acquire only a fraction of the nodes available, and the nodes allocated do not correspond to any standard topology, even when the machine does. For example, we show below in figure 1.1 an allocation for 1000 nodes on the 3D torus Jaguar machine at ORNL. We can see that the nodes allocated are several discontinuous pieces of the larger machine. Assignment of tasks to nodes that take this into account can reduce communication overhead.

The latest massively parallel systems are predominantly being built from nodes with multi-cores. This trend is evident when considering the systems of the Top500 list of supercomputers [17], which are expected to feature, in a close future, fat many-core nodes composed of as many

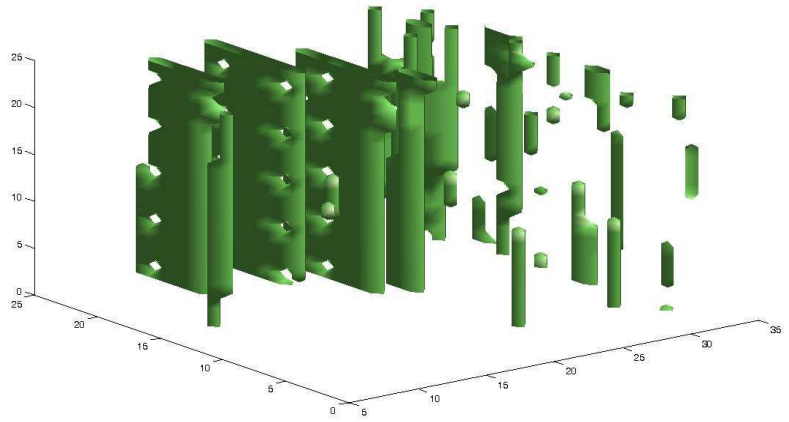

Figure 1.1: Allocation of 1000 nodes on Jaguar; The axes correspond to indices on the 3D torus, and the Green region corresponds to the allocated nodes.

as a hundred of cores. As the number of processor cores integrated onto a single die increases, varieties of ways of interconnecting these on-chip cores are explored. Buses, rings and 2D meshes have been used as the on-chip interconnection network topologies. The increasing number of cores on the chip underscores the importance of on-chip communication performance and hence the mapping on the chip. We evaluate the mapping within a multi-core using the IBM Cell processor as a case study.

1.2 Research Goals

The first part of this thesis discusses the methodology adopted for efficient mapping on the Cell processor. The Cell based blades are used in the Roadrunner supercomputer at LANL, one of the first supercomputer to reach the petaflop mark. The bulk of the

computational workload on the Cell processor is carried by eight co-processors called SPEs. The SPEs are connected to each other and to main memory by a high speed bus called the Element Interconnect Bus (EIB). The bandwidth utilization on EIB is reduced due to the congestion created by the simultaneous communications. We observed that the effective bandwidth obtained for inter-SPE communication is strongly influenced by the assignment of threads to SPEs (Thread-SPE affinity). We have demonstrated a performance improvement of around 10%-12% for a communication intensive Monte Carlo particle simulation application.

The next part of the thesis discusses the techniques used for determining the mapping of parallel tasks to nodes in massively parallel machines so as to efficiently utilize the networking infrastructure. We have used a topology aware mapping scheme to improve the performance of communication in the load balancing step of a QMC application on Jaguar. In this scheme, we obtained the physical topology of Jaguar (which implies the routing scheme) from the system administrator. Using this information, we rearrange the order of the MPI processes by creating a new communicator that ranks the nodes according to their relative position on a space-filling curve: this ensures that each node is likely to send data to nearby nodes in the communications in this phase. Using this reordering, we were able to reduce the communication time in the load balancing phase by 30% on 120,000 cores and 20% on 12,000 cores, and this mapping also reduced MPI_allgather time by a similar amount.

Encouraged by these results, we then embarked on the task of finding more generic solutions to this mapping problem. The optimal mapping

of processes to nodes is an NP hard problem, and hence heuristics are used to solve it approximately. Recent works use heuristics that can be intuitively expected to reduce network congestion and then evaluate them either empirically or using some metric. The hop-byte metric has attracted much attention recently as metric to evaluate the quality of a mapping. It is defined as the sum over all the messages of the product of the message size and number of hops the message has to traverse. On a store-and-forward network, this would correspond to the total communication volume. The intuition behind this metric is that if the total communication volume is high, then it is also likely to increase the contention for specific links, which would then become communication bottlenecks. Although this metric does not directly measure the communication bottleneck caused by contention, heuristics with low values of this metric tend to have smaller communication overheads. This serves as a justification for this metric. The advantage of this metric is that it requires only the machine topology, while computing contention would require routing information.

In contrast to other approaches, we use the hop-byte metric for producing the mappings too, rather than just using it for evaluating the mapping. Optimizing for the hop-byte metric can easily be shown to be a specific case of the Quadratic Assignment Problem (QAP), which is NP hard. Exact solutions can be determined using branch and bound for small problem sizes. We use the existing GRASP heuristic for medium-sized problem. An advantage of the QAP formulation is that we can use theoretical lower bounds to judge the quality of our solution. The time taken to determine the mapping using an exact solver or GRASP increases rapidly with problem size. In that case, we consider a couple

of alternate approaches. In the first case, we develop new heuristics that improve on some limitations of other heuristics for this problem. In the second case, we use graph partitioning to break up the problem into smaller pieces, and then apply GRASP to each partition.

We evaluate our approach on six different communication patterns. We determine the values of the metric for different heuristics using node allocations obtained on the Kraken supercomputer at NICS. In contrast to other works that usually assume some standard topology, our results are based on actual allocations obtained. We see up to 75% reduction in hop-bytes over the default allocation, and up to 66% reduction over existing heuristics. Furthermore, our results are usually within a factor of two of a theoretical lower bound on the solution. For small problem sizes, that lower bound is usually just a little over half the exact solution. Consequently, it is likely that our solutions are close to optimal.

Though the above techniques optimize the hop-byte metric very well, this theoretical improvement is not as well reflected in the empirical results using this optimized metric. This highlights the limitations of the hop-byte metric, and hence we considered the maximum contention metric as an alternative. A heuristic to optimize for this metric shows promising empirical performance.

1.3 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 discusses previous work in this field. There was much work in the 1980s on topology aware mapping when the systems used to have packet switched networks. In packet-switching networks, communication performance is sensitive to path lengths and minimizing dilation costs is usually

the mapping objective [18, 19]. However, experimental results confirm that in wormhole-routed networks, communication performance does not depend any more on path lengths and minimizing dilation costs is no longer a concern [20] and hence the research in this area ceased. There has been recent work in the past seven years triggered by the development of IBM Blue Gene/L, a supercomputer using 3D torus topology. Contributions of this thesis and differences with the recent work are also discussed. In chapter 3, we provide a brief survey of the existing interconnect topologies and routing schemes used in the multi-cores and in the large machines. This chapter also presents architectural details of the machines used for experiments in this thesis. Topology information of an allocated job is a vital for mapping algorithms and the process of topology discovery on Cray machines is discussed.

Chapter 4 presents the mapping algorithms and performance results on a Cell processor. The evaluation of the mapping techniques are done with some standard communication patterns as well with a realistic application. The chapter concludes with a description of a tool which can be used to automatically determine the optimal mapping for any given communication pattern. The rest of the thesis discusses the mapping techniques used for the large machines.

Chapter 5 demonstrates the performance improvements of mapping for a specific production scale scientific application QWalk [21]. The load balancing step can be a significant factor affecting performance, and we propose a new dynamic load balancing algorithm and evaluate it theoretically and empirically. Empirical results on the petaflop Cray XT Jaguar supercomputer showed up to 30% improvement in performance on 120,000 cores. These techniques used for mapping are specific to

the QWalk type applications, and a more general approach to mapping is required and that will be the topic of the next chapter. Chapter 6 discusses the methods used to optimize the hop-byte metric, an important contribution of this thesis. Apart from proposing two new heuristics for mapping, whose quality with respect to the metric value is evaluated, we also use a heuristic to directly optimize for this metric. The scalability of the heuristics is analyzed and subsequently improved using graph partitioning techniques. Then we also present results with the maximum contention metric in Chapter 7. Empirical results show that this metric better represents the mapping quality compared to the hop-byte metric. We then present a summary and new directions for research in the concluding chapter.

2. Related Work

Mapping of processes to nodes based on network topology attracted much attention in the earlier years of parallel computing. Techniques to map processes onto various high-performance interconnects such as crossbar based network, hypercube network, switch-based interconnects, and multidimensional grids were developed [6], [7], [8]. The research in this area lost its importance for some time with the advent of communication mechanisms such as worm-hole routing. However, for reasons explained in section 1.1, it had once again attracted much attention recently.

The idea of the mapping problem is that given an application communication graph and the processor allocation graph, the objective is to map the first graph onto the second graph such that most messages pass through a small number of links in the network. This problem is similar to the graph embedding problem which is NP-hard, and hence heuristics are used to arrive at a reasonable solution. The application communication graph is either regular or irregular. And the node allocation graph on most of the systems is arbitrary, except on BlueGene systems where contiguous cuboidal partitions are provided for the jobs. However, much of the work in the recent past on topology aware mapping assumes the node allocation graph as regular. Our attempt

in this thesis is to map effectively onto the more realistic arbitrary node allocations. In this chapter, we first discuss the research done on mapping communication graphs onto regular allocation graphs. We then present the work done on mapping onto arbitrary node allocations. We also discuss few mapping techniques that are specific to some applications or specific to some network topology.

2.1 Mapping onto Regular Graphs

Different topological strategies for mapping torus process topologies onto the torus network of Blue Gene/L were presented by Yu, Chung, and Moreira [4]. Their work deals with node mappings for simple regular graphs such as 1D rings, 2D meshes and 3D grids/tori. Several techniques for mapping more generic regular communication graphs onto regular topologies were developed [15]. They developed a framework to automatically map 2D communication graphs onto 2D/3D mesh and torus topologies. Both these works use the hop-byes as the metric to evaluate the mapping quality. The mapping is a NP-hard problem [28]; hence heuristics are used to approximate the optimal solution. Heuristic techniques for mapping applications with irregular communication graphs to mesh and torus topologies were developed, and some of them even take advantage of the physical coordinate configuration of the allocated nodes [12]. The performance of these heuristics were evaluated based on the hop-byte metric.

2.2 Mapping onto Irregular Graphs

The more realistic and difficult problem is to map onto the arbitrary node allocation graphs. Hoefler and Snir [13] present mapping algorithms that are meant for more generic use and the algorithms are evaluated using the maximum congestion metric – the message volume on the link with maximum congestion. Their heuristics based on recursive bisection and graph similarity were used to map application communication patterns on realistic topologies. The metrics here again are used to evaluate the mappings rather than being used to determining the mapping. The algorithm Greedy Graph Embedding (GGE) proposed in [13] is used by us for comparison, because it performs best among the heuristics they have proposed. Furthermore, the algorithm can be used with arbitrary communication patterns and network topologies, even though the implementation in [13] was more restricted. Graph partitioning libraries such as SCOTCH [26] and Metis [27] provide support for mapping graphs to network topologies.

2.3 Application Specific Mapping

It is beneficial to use application specific features to assist the mapping algorithm in arriving at a better solution. Bhatele and Kale [11] proposed topology-aware load-balancing strategies for Molecular Dynamics applications using CHARM++. Their analysis maps mesh and torus process topologies to other mesh and torus network topologies. Several applications [22], [23], [24] have developed their own hand-tuned mapping algorithms and mechanisms to optimize communication.

2.4 Mapping for Specific System Topologies

Krishna et al. developed topology aware collective algorithms for Infiniband networks. These networks are hierarchical with multiple levels of switches, and this knowledge was used in designing efficient MPI collective algorithms [16]. The reduced scalability of the latency and effective bandwidth due to interconnect hot spot for fat-tree topologies is addressed with topology-aware MPI node ordering and routing-aware MPI collective operations [25].

2.5 Topology Aware Job Scheduling

Massively parallel systems such as Cray XT and Blue Gene/P systems are generally heavily loaded with multiple jobs running and sharing the network resources simultaneously, this results in application performance being dependent on the node allocation for a particular job. Balaji et al. [29] analyzed the impact of different process mappings on application performance on a massive Blue Gene/P system [10]. They show that the difference can be around 30% for some applications and can even be two fold for some. They have developed a scheme whereby the user can describe the application communication pattern before running a job, and the runtime system then provides a mapping that potentially reduces contention.

2.6 Contributions of This Thesis

Here we highlight the main contributions this thesis. The first major contribution is the work on mapping effects on a multi-core

processor. Ours is one of the first work to show the effects of affinity on a Cell processor. A tool to automatically detect the optimal mapping is developed, which eases burden on the application programmers. This work created an awareness about the importance of mapping on multi-core processors. Though Cell is no longer used now, the ideas presented in this work can well be applied for generating mapping techniques aimed for other multi-core processors.

The massively parallel machines will generally have a regular interconnect topology, however, that does not guarantee that the nodes allocated for a specific job allocation follow a regular topology. Many of the previous works on the mapping assumed a regular node allocation topology, and the mapping techniques and heuristics are designed accordingly. However, the mapping heuristics proposed in this work do not have any such assumptions and hence they are more applicable for the realistic node allocation scenarios.

The problem is motivated by showing some encouraging performance improvements on a production scale application, QWalk. We also developed a new dynamic load balancing algorithm which is more efficient theoretically and empirically, especially for future machines. To evolve more generic mapping techniques, we have posed the mapping problem as a well know QAP problem and used a heuristic to optimize the metric itself rather than just using it for evaluation. This heuristic using for communication graphs from production codes and allocations on real machines showed encouraging results. An alternate metric, maximum contention metric, to better quantify the mapping quality is also used and evaluated.

3. Interconnection Topologies

In the area of high performance computing, the interconnect is second only to the microprocessor in terms of performance-critical components. As systems expand into ever increasing number of processors, the interconnect is the enabling technology that allows the underlying computation to scale efficiently. Topology and routing are the two key properties of a network.

3.1 Topology and Routing

Topology of a network determines the arrangement of nodes and links in the network and it is one of the first step in a network design. Several different topologies are used in the largest supercomputers today. Three dimensional tori and meshes (Cray's XK and XT series, and IBM Blue Gene series) and fat-trees (Infiniband and Federation) are the most commonly used topologies. The next important property of network is routing and it determines path(s) from source to destination. A routing algorithm's ability to balance traffic (or load) has a direct impact on the throughput and performance of the network. A routing method that does not use the state of the network for routing decisions is termed oblivious routing. And the method that uses the state of the network is called

adaptive routing. IBM Blue Gene/P uses minimal adaptive routing and the Cray supercomputers generally use dimension order routing.

Apart from the growing importance of networking used to connect the nodes in a big machine, there has been a growing importance of networking used within a node. All the machines in the current Top500 list has multi-core processor(s) in their nodes. As the number of on-chip cores on a multi-core increases, a scalable and high-bandwidth communication fabric to connect them becomes critically important. This evolution of interconnection networks as core count increases is clearly illustrated in the choice of a flat crossbar interconnect connecting all eight cores in the Sun Niagara (2005) [30], four packet-switched rings in the 9-core Cell (2005) [31], five packet-switched meshes in the 64-core Tilera TILE64 (2007), and Intel's SCC [32] and Core i7 [33].

3.2 Categorization of Topologies

A network topology can be categorized as direct or indirect. In a direct topology, each terminal node (e.g. a processor core in a chip multiprocessor or a compute node in supercomputer) also acts as a routing element, so all the nodes are sources and destinations of traffic. In a direct topology, nodes can source and sink traffic, as well as switch through traffic from other nodes. In an indirect topology, the routing elements are distinct from terminal nodes; only terminal nodes are sources and destinations of traffic, intermediate nodes simply switch traffic to and from terminal nodes. Most of the on-chip networks built till date have used direct topologies.

3.2.1 Indirect Topologies

A fat-tree [34] is logically a binary tree network, with processing nodes at the leaves and switches at the intermediate nodes. In a fat-tree, the wiring resources increase for stages closer to the root node. Figure 3.1 depicts a simple three level binary fat-tree.

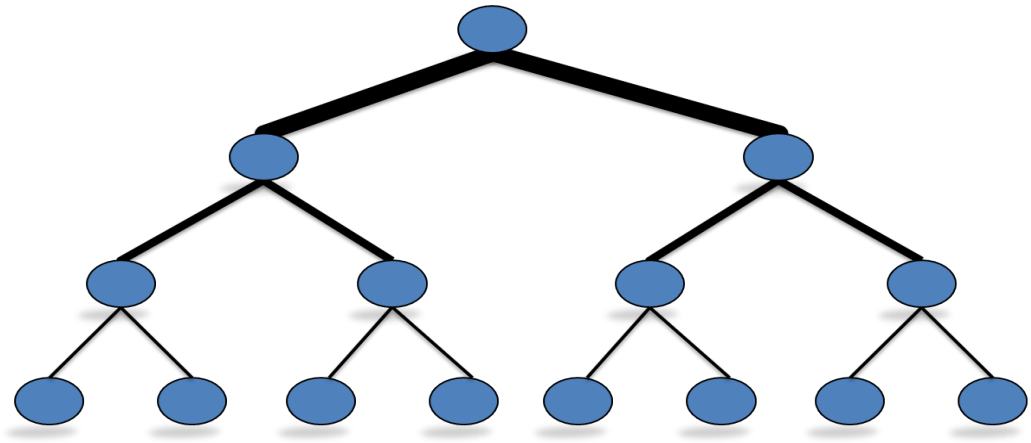

Figure 3.1: Fat-tree topology.

Fat-tree or CLOS topologies are well suited for smaller node-count systems. Fat-tree topologies provide non-blocking interface and small hop counts resulting in reasonable latency for MPI jobs. At the same time, fat-tree topologies do not scale linearly with cluster size. As cluster size grows, cabling and switching become increasingly difficult and expensive with very large core switches required for larger clusters. Stampede machine at TACC has more than 6,000 nodes interconnected via a fat-tree, FDR (fourteen data rate) InfiniBand interconnect. The Ranger machine at Texas Advanced Computing Center (TACC) has a 7-stage fat-tree, with 3,936 compute nodes connected with two 3,456 port SDR Sun Infiniband datacenter switches.

As more the system grows, more are the limits of the fat-tree switched

topologies in terms of cost, maintainability, power consumption, reliability and, above all, scalability.

3.2.2 Direct Topologies

Hypercube topology has many useful features such as small diameter, high connectivity, symmetry and simple routing. However, lack of scalability is its major drawback, which limits its use in building large size systems in the current generation. Mesh and torus networks, a generalization of the hypercube, can be described as k-ary n-cubes, where k is the number of nodes along each dimension, and n is the number of dimensions. For instance, a 6 x 6 mesh or torus is a 6-ary 2-cube with 36 nodes, a 8 x 8 mesh or torus is a 8-ary 2-cube with 64 nodes, while a 4 x 4 x 4 mesh or torus is a 4-ary 3-cube with 64 nodes. This notation assumes the same number of nodes on each dimension, so total number of nodes is 'kn'. To map well to the planar substrate, most on-chip networks utilize 2D topologies; this is not the case for off-chip networks where cables between chassis provide 3D connectivity. In each dimension, k nodes are connected with channels to their nearest neighbors. Ring topologies fall into the torus family of network topologies as k-ary 1-cubes. Figure 3.2 shows a 3D mesh network, a 2,3,4-ary 3-mesh.

The basic shortest-path routing in general meshes is simple. It is called dimension ordered routing. The addresses of the sender and receiver are inspected in a specified order and the message is sent via the first port corresponding to a coordinate in which the addresses differ. In 2D and 3D meshes, this is called XY and XYZ routing. Cyclic structure of tori makes shortest path routing algorithms more complicated, but the

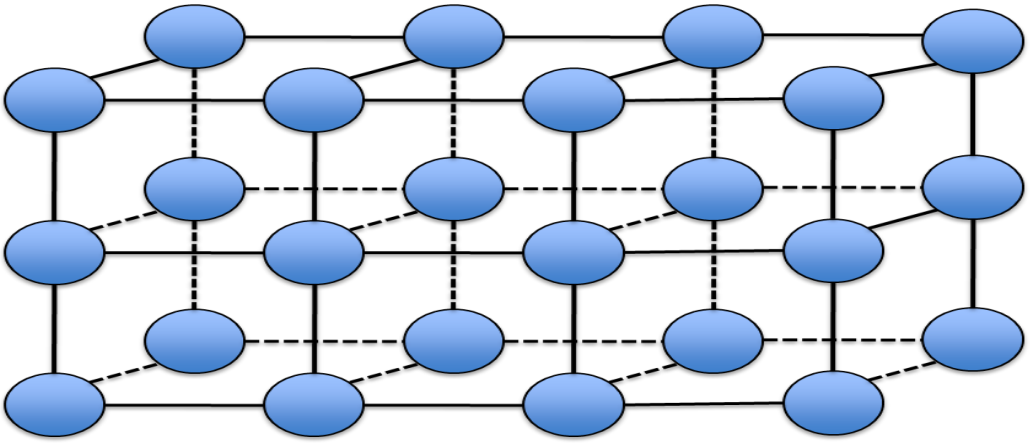

Figure 3.2: A three dimensional Mesh

basic approach remains dimension ordered routing.

The diameter of graph (G) is defined as the maximum distance between any two vertices of G. For a 3D mesh, if the size of the mesh in each dimension is n, the diameter of the mesh is 3 x (n - 1). For a 3D torus with size n in each dimension, the diameter of the torus is 3 x (n/2)

Practicality of Torus Topologies

Even though torus topologies are not asymptotically scalable, because of simplicity of design and other practical considerations, torus networks are a popular choice for modern day supercomputers. As per the Top500 list released in June 2012, six of the ten fastest machines use a 3D torus interconnect topology. This thesis focuses primarily on such networks for the experiments, though the results, especially optimization of hop-byte and maximum contention metrics can very well be applied to other topologies. We now describe the configurations of multi-core processors as well as supercomputers that use these direct topologies.

3.3 On-chip Interconnection Topologies

The interconnect of Cell processor consists of four unidirectional rings, two in each direction. As each ring is 16 bytes wide, runs at 1.6 GHz, and can support 3 concurrent transfers with a total network bisection bandwidth of 204.8 GB/s [37, 41]. However, the bus access semantics and the ring topology can lead to a worst-case throughput of 50% or even less with adversarial traffic patterns.

The Intel TeraFLOPS chip consists of an 8 x 10 mesh, with each channel composed of two 38-bit unidirectional links. It runs at an aggressive clock speed of 5 GHz on a 65 nm process. This design gives it a bisection bandwidth of 380 GB/s or 320 GB/s of actual data bisection bandwidth, since 32 out of the 38 bits of a flit are data bits; the remaining 6 bits are used for sideband. Again, depending on the traffic, routing and flow control, realizable throughput will be a fraction of that [35].

Intel Nehalem EX use high-speed ring topologies for inter core communications. It uses twin 256-bit wide rings encircling eight cores for bi-directional interprocessor communications. For the 32-core prototype chips, the predecessor to the 50-core Knight's Corner boards, Intel has boosted its ring topology to 1024-bits wide, offering bi-directional 512-bit wide rings.

3.4 Supercomputer Interconnection Topologies

3.4.1 Jaguar

The multi-petaflop system at ORNL, called Jaguar, prior to the recent upgrade to Titan, housed 18,688 Cray XT5 compute nodes

interconnected with SeaStar in a 3D torus topology. In recent transition of Jaguar to Titan, it now uses Cray's Gemini interconnect. The massively parallel system at NICS, Kraken is a Cray XT5 system with 9,408 nodes interconnected with the SeaStar router through HyperTransport. The SeaStars are all interconnected in a 3D torus topology. The routing on these machines uses fixed paths between pairs of nodes, and sends data along the x coordinate of the torus, in the direction of the shortest distance, then in the y direction, and finally in the z direction. Thus, in a fault-free network, this straightforward dimension-ordered routing (DOR) algorithm will enable balanced traffic across the network links.

3.4.2 Blue Waters

Blue Waters supercomputer coming up at NCSA is made up of over 300 cabinets, with 25,000 nodes, over 380,000 cores, aided by more than 3,000 NVIDIA GPU's. It employs the Cray Gemini interconnect, which implements a 3D torus topology (23x24x24) with a bisection bandwidth of 10.35 TB/s. Fujitsu's Tofu [36] interconnect is used in Japan's K supercomputer, the system ranked 2 in the Top500 list of June 2012. Tofu realizes scalable systems beyond 100,000 nodes with low power consumption, low latency, and high bandwidth. The Tofu interconnect uses a 6D mesh/torus topology in which each cubic fragment of the network has the embeddability of a 3D torus graph, allowing users to run multiple topology-aware applications [36].

3.4.3 Blue Gene/Q

In a Blue Gene/Q based system, compute nodes are interconnected via a 5D torus. To support a 5D torus, 10 bidirectional ports, or links, are

required. With an internal 5D torus interconnect supporting 563 GB/s of bisection bandwidth and a memory bandwidth of 32 GB/s, each BG/Q processor can deliver 204.8 GFlop/s of peak computing power. With 1024 16-core compute chips, a BG/Q rack can deliver 209 TFlop/s of peak computing performance. The largest deployment of BG/Q, the Sequoia system in the LLNL is composed of 96 BG/Q racks containing 1,572,864 cores exhibiting an overall peak power of around 16 PFlop/s. The other versions of Blue Gene, Blue Gene/L and Blue Gene/P also use the 3D torus topology.

3.5 Comparision of Direct and Indirect Topologies

Most commonly used topology for InfiniBand fabrics today is a fat-tree topology. With a fat-tree topology, every node has equal access bandwidth to every other node in the cluster. Fat-tree is a great topology for running large scale applications where nodes do a lot of communication with each other. In contrast, a 3D torus topology is best used for applications that use communications between localized compute nodes, as this locality is usually a requirement to achieve optimal performance with a torus.

Though the SDSC Gordon supercomputer uses the Infiniband interconnect, the topology it employs is not a fat-tree. Because of the nature of the applications targeted for the system, where in many cases localization constraints within the application are employed, the 3D torus architecture is more suitable from a performance standpoint. The 3D torus in Gordon was built using 36-port switch nodes in a 4x4x4 configuration, for a total of 64 torus junctions. Each of these junctions connects to 16 compute nodes, and 2 IO nodes, with inter-node links

using 3 switch ports in each of the +/- X, Y, and Z directions. It can be summarized that fat-tree topology provides the best performance solution, however, 3D torus can be more cost effective, easier to scale, good fit for applications with locality.

3.6 Network Performance Metrics

To characterize the performance of the network, abstract metrics such as hop count and maximum channel load can be used. Assuming ideal routing, hop count can be used as a metric to evaluate the performance of a network. Hop count is defined as the number of hops a message takes from source to destination. The diameter of a network is determined as the largest minimum hop count in the network. The average hop count for a torus network is found by averaging the minimum distance between all possible node pairs. The other metric, maximum channel load, is based on the intuition that the most congested link for a given communication pattern will limit the overall network bandwidth. Bottleneck channels determine the saturation throughput. Higher the channel load, greater the chance of network congestion. For distance sensitive routing such as store-and-forward, small average hop distance allows small communication latency. For distance insensitive routing such as wormhole routing, short distances imply less used links and buffers, and therefore less communication contention, it is also crucial for them.

3.7 Topology Detection

The other important issue is finding the topology of the nodes allocated for a job, though the entire machine has a regular topology, the allocation topology is irregular often, as the available nodes chosen by the job scheduler could potentially be taken from different parts of system. Topology information of an allocated job is a vital for mapping algorithms and the topology discovery on Cray machines can be done using Cray Resiliency Communication Agent (RCA) library [38]. Detecting topology for an Infiniband based network is non trivial and a tool for doing the same is developed recently using the neighbor joining algorithm [39].

4. Optimizing Assignment of Threads to SPEs on Cell BE

4.1 Introduction

The Cell BE contains a PowerPC core, called the PPE, and eight co-processors, called SPEs. The SPEs are meant to handle the bulk of the computational workload, and have a combined peak speed of 204.8 GFlop/s in single precision and 14.64 GFlop/s in double precision. They are connected to each other and to main memory by a high speed bus called the EIB, which has a bandwidth of 204.8 GB/s.

However, access to main memory is limited by the memory interface controller's performance to 25.6 GB/s total (both directions combined). If all eight SPEs access main memory simultaneously, then each sustains bandwidth less than 4 GB/s. On the other hand, each SPE is capable of simultaneously sending and receiving data at 25.6 GB/s in each direction. Latency for inter-SPE communication is under 100 ns for short messages, while it is a factor of two greater to that of main memory. It is, therefore, advantageous for algorithms to be structured such that SPEs tend to communicate more between themselves, and make less use of main memory. The latency between each pair of SPEs is identical

for short messages and so affinity does not matter in this case. In the absence of contention for the EIB, the bandwidth between each of pair of SPEs is identical for long messages too, and reaches the theoretical limit. However, we show later that in the presence of contention, the bandwidth can fall well short of the theoretical limit, even when the EIB's bandwidth is not saturated. This happens when the message size is greater than 16 KB. It is, therefore, important to assign threads to SPEs to avoid contention, in order to maximize the bandwidth for the communication pattern of the application.

The outline of the rest of the chapter is as follows. In section 4.2, we summarize important architectural features of the Cell processor relevant to this work. We next show that thread-SPE affinity can have significant influence on inter-SPE communication patterns in section 4.3. We also identify factors responsible for reduced performance. We use these results to suggest good affinities for common communication patterns. We then evaluate them empirically in section 4.4. We next discuss optimizing affinity on the Cell Blade in section 4.5. A tool that can automatically suggest optimal mapping, and its evaluation using few standard communication patterns and a realistic application will be described next.

4.2 Cell Architecture Overview

We summarize below the architectural features of the Cell of relevance to this work, concentrating on the communication architecture. Further details can be found in [41, 44].

Figure 4.1 provides an overview of the Cell processor. It contains a cache-coherent PowerPC core called the PPE, and eight co-processors,

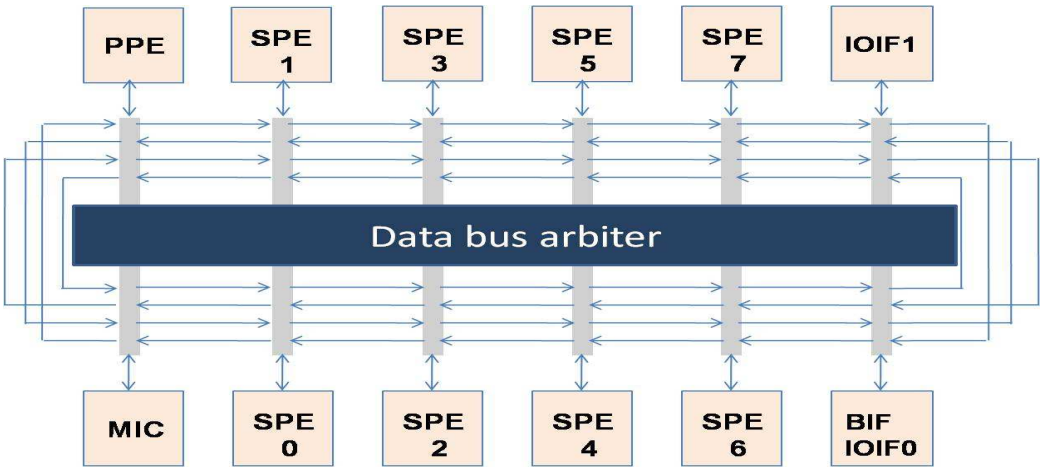

Figure 4.1: Overview of the Cell communication architecture.

called SPEs, running at 3.2 GHz each. It has a 512 MB - 2 GB external main memory. An XDR memory controller provides access to main memory at 25.6 GB/s total, in both directions combined. The PPE, SPE, and memory controller are connected via the EIB. The maximum bandwidth of the EIB is 204.8 GB/s. In a Cell Blade, two Cell processors communicate over a BIF bus. The numbering of SPEs on processor 1 is similar, except that we add 8 to the rank for each SPE.

The SPEs have only 256 KB local store each, and they can directly operate only on this. They have to explicitly fetch data from memory through DMA in order to use it. Each SPE can have 16 outstanding requests in its DMA queue. Each DMA can be for at most 16 KB. However, a DMA list command can be used to scatter or gather a larger amount of data. When we perform experiments on messages larger than 16 KB, we make multiple non-blocking DMA requests, with total size equal to the desired message size, and then wait for all of them to complete.

In order to use an SPE, a process running on the PPE spawns an SPE thread. Each SPE can run one thread, and that thread accesses data

from its local store. The SPEs' local stores and registers are mapped to the effective address space of the process that spawned the SPE threads. SPEs can use these effective addresses to DMA data from or to another SPE. As mentioned earlier, data can be transferred much faster between SPEs than between SPE and main memory [41, 44].

The data transfer time between each pair of SPEs is independent of the positions of the SPEs, if there is no other communication taking place simultaneously [42]. However, when many simultaneous messages are being transferred, transfers to certain SPEs may not yield optimal bandwidth, even when the EIB has sufficient bandwidth available to accommodate all messages.

In order to explain this phenomenon, we now present further details on the EIB. The EIB contains four rings, two running clockwise and two running counter-clockwise. All rings have identical bandwidths. Each ring can simultaneously support three data transfers, provided that the paths for these transfers don't overlap. The EIB data bus arbiter handles a data transfer request and assigns a suitable ring to a request. When a message is transferred between two SPEs, the arbiter provides it a ring in the direction of the shortest path. For example, transfer of data from SPE 1 to SPE 5 would take a ring that goes clockwise, while a transfer from SPE 4 to SPE 5 would use a ring that goes counter-clockwise. If the distances in clockwise and anti-clockwise directions are identical, then the message can take either direction, which may not necessarily be the best direction to take, in the presence of contention. From these details of the EIB, we can expect that certain combinations of affinity and communication patterns can cause non-optimal utilization of the EIB. We will study this in greater detail below.

4.3 Influence of Thread-SPE Affinity on Inter-SPE Communication Performance

We show that the affinity significantly influences communication performance when there is contention. We then identify factors that lead to loss in performance, which in turn enables us to develop good affinity schemes for a specified communication pattern.

4.3.1 Experimental Setup

The experiments were performed on the CellBuzz cluster at the Georgia Tech STI Center for Competence for the Cell BE. It consists of Cell BE QS20 dual-Cell Blades running at 3.2 GHz with 512 MB memory per processor. The codes were compiled with the ppuxlc and spuxlc compilers, using the -O3 -qstrict flags. SDK 2.1 was used. Timing was performed by using the decrementer register which runs at 14.318 MHz, yielding a granularity of around 70 ns.

4.3.2 Influence of Affinity

We mentioned earlier that affinity does not matter when there is no contention. Figure 4.2 presents performance results when there are several messages simultaneously in transit, for the following communication pattern: threads *ranked* i and $i+1$ exchange data with each other, for even i. This is a common communication pattern, occurring in the first phase of Recursive Doubling algorithms, which are used in a variety of collective communication operations. The results are presented for three specific affinities and for the default. (The default

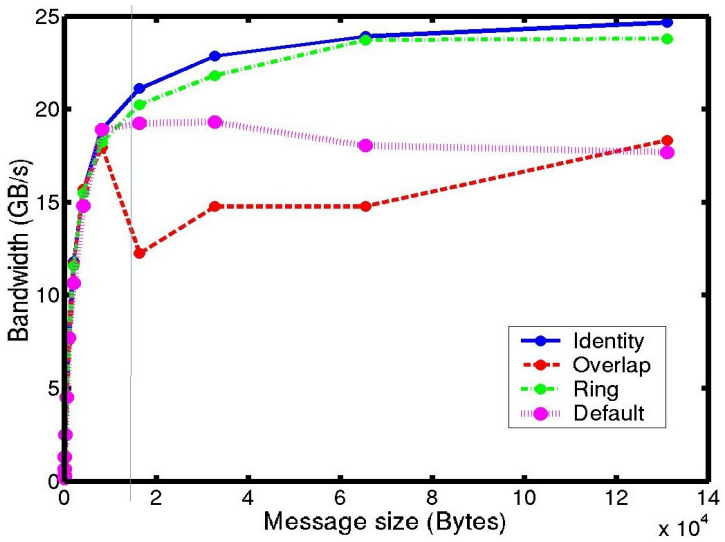

Figure 4.2: Performance in the first phase of Recursive Doubling – minimum bandwidth vs. message size.

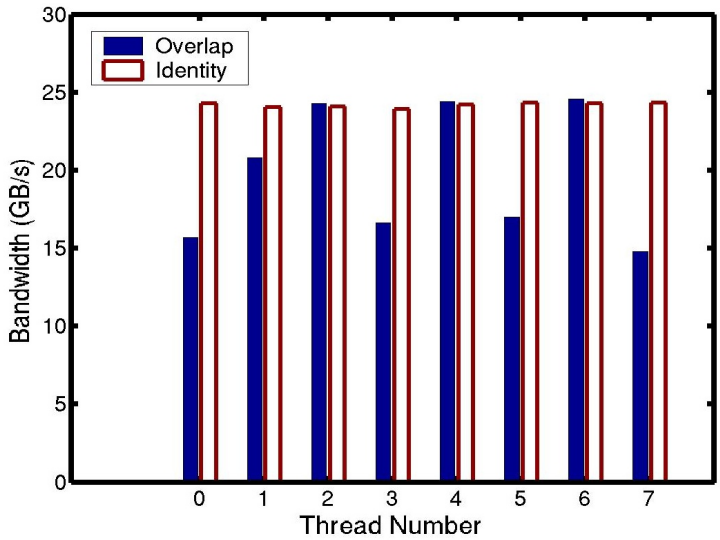

Figure 4.3: Performance in the first phase of Recursive Doubling – bandwidth on different SPEs for messages of size 64 KB.

affinity is somewhat random as mentioned earlier; we present results for one specific affinity returned by default – a more detailed analysis is presented later.) We provide details on the affinities used in a later section (4.4). Our aim in this part of the section is just to show that affinity influences performance.

Figure 4.2 is meant to identify the message size at which affinity starts to matter. Note that the bandwidths obtained by different SPEs differ. The results show the minimum of those bandwidths. This is an important metric because, typically, the speed of the whole application is limited by the performance of the slowest processor. We can see that the affinity does not matter at less than 16 KB data size, because the network bandwidth is not fully utilized then. With large messages, as a greater fraction of the the network bandwidth is utilized, affinity starts having a significant effect. Figure 4.3 shows the bandwidth obtained by each SPE with two different affinities, in one particular trial. We can see that some SPEs get good performance, while others perform poorly with a bad affinity.

We next give statistics from several trials. Figure 4.4 gives the results for the default affinity. If we compare it with fig. 4.6, we can see that the standard deviation is much higher for the default affinity than for the Identity affinity (which is described later). There are two possible reasons for this: (i) In different trials, different affinities are produced by default, which can lead to a large variance. (ii) For the same affinity, the variance could be high. It turns out that in these eight trials, only two different affinities were produced. The mean and standard deviation for each affinity was roughly the same. The high variance is primarily caused by the inherent variability in each affinity. This often happens

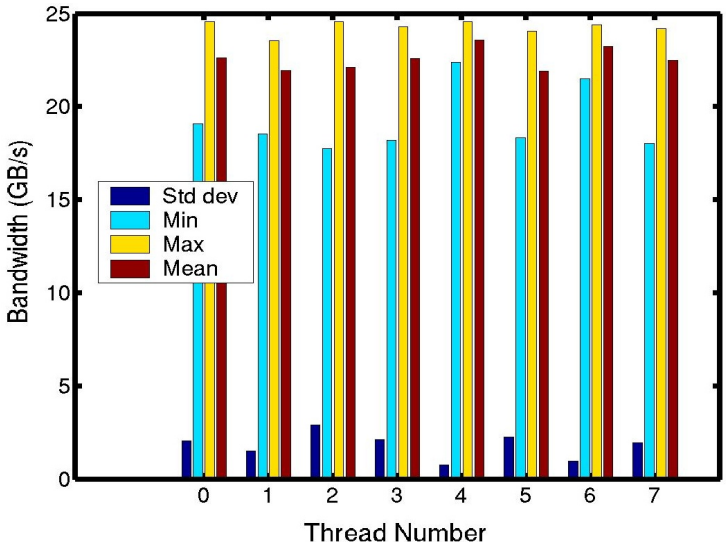

Figure 4.4: Performance in the first phase of Recursive Doubling with default affinities – bandwidth on each thread.

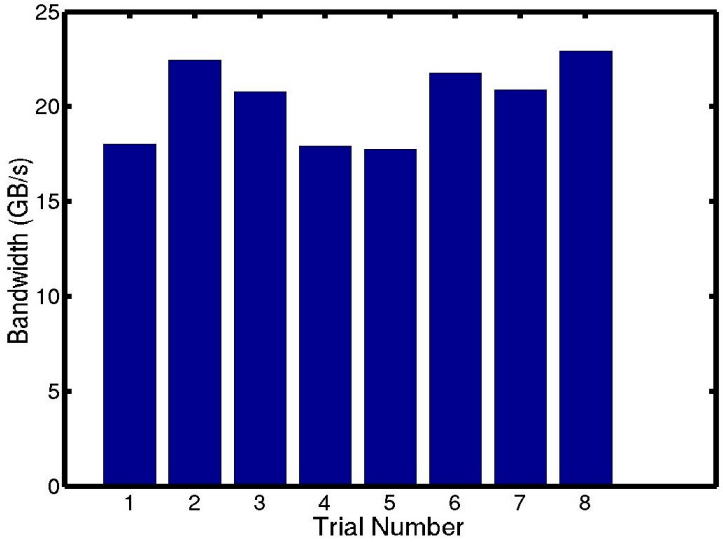

Figure 4.5: Performance in the first phase of Recursive Doubling with default affinities – minimum bandwidth in each of the eight trials.

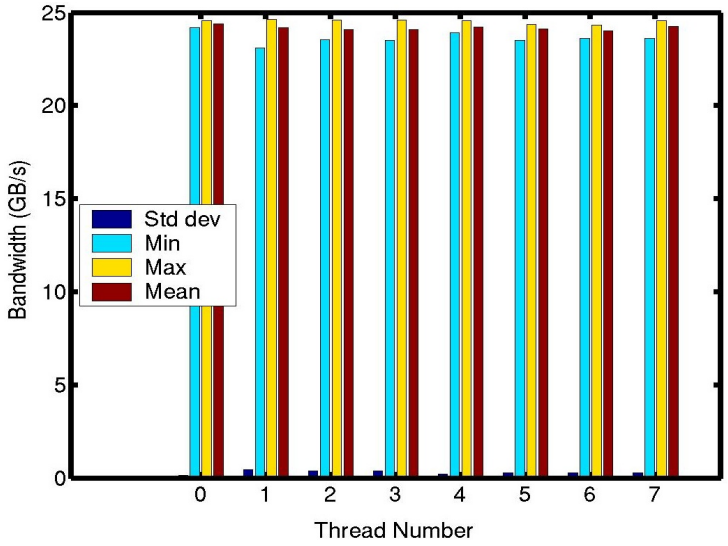

Figure 4.6: Performance in the first phase of Recursive Doubling with the Identity affinity – bandwidth on each thread.

in cases where contention degrades performance – when there is much contention, there is more randomness in performance which causes some SPEs to be have lower performance. Figure 4.4 shows that most of the SPEs have good mean performance. However, we also observe that most SPEs have a low value of their worst performance. In most trials, some thread or the other has poor performance, which proves to be a bottleneck. In contrast, all SPEs consistently obtain good performance with the Identity affinity. This contrast is illustrated in figures 4.5 and 4.7. For some other communication patterns, the default assignment yields some affinities that give good performance and some that yield poor performance. In those cases the variance is more due to the difference in affinities.

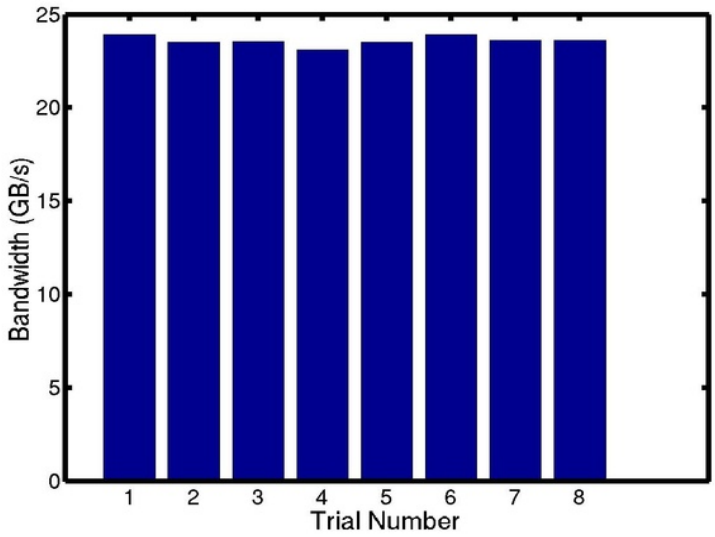

Figure 4.7: Performance in the first phase of Recursive Doubling with the Identity affinity – minimum bandwidth in each of the eight trials.

4.3.3 Performance Bottlenecks

We experimented with simple communication steps, in order to identify factors that are responsible for loss in performance. These results, which are presented in [42], suggest the following rules of thumb for large messages.

1. Avoid overlapping paths for more than two messages in the same direction. This is the most important observation.

2. Given the above constraints, minimize the number of messages in any direction by balancing the number of messages in both directions.

3. Don't make any assumptions regarding the direction of transfer for messages that travel half-way across the EIB ring.

4.4 Affinities and Their Evaluation

We now propose some affinities that appear reasonable, and evaluate their performance empirically. We used the affinities mentioned below, other than the default. They were designed to avoid the bottlenecks mentioned in section 4.3.

- *Identity* The thread ID is identical to the physical ID of the SPE.
- *Ring* (Physical ID, Thread Number) mapping: {(0, 0), (1, 7), (2, 1), (3, 6), (4, 2), (5, 5), (6, 3), (7, 4)}. Thread ranks that are adjacent are also physically adjacent on the EIB ring, and thread 7 is physically adjacent to thread 0.
- *Overlap* Mapping: {(0, 0), (1, 7), (2, 2), (3, 5), (4, 4), (5, 3), (6, 6), (7, 1)}. Here, threads with adjacent ranks are half way across the ring. We would expect poor results on a ring communication pattern. We use this as a lower bound on the performance for a Ring communication pattern.
- *EvenOdd* Mapping: {(0, 0), (1, 4), (2, 2), (3, 6), (4, 1), (5, 5), (6, 3), (7, 7)}. Even ranks are on the left hand side and odd ranks are on the right hand side. This affinity was designed to perform well with recursive doubling.
- *Leap2* Mapping: {(0, 0), (1, 4), (2, 7), (3, 3), (4, 1), (5, 5), (6, 6), (7, 2)}. This deals with a limitation of the Ring affinity on the Ring communication pattern. The Ring affinity causes all communication to be in the same direction with that communication pattern, causing unbalanced load. The Leap2

Pattern	Name
1. $0 \leftarrow 7, 1 \leftarrow 0, 2 \leftarrow 1, ..., 7 \leftarrow 6$	Ring
2. $0 \leftrightarrow 1, 2 \leftrightarrow 3, 4 \leftrightarrow 5, 6 \leftrightarrow 7$	Recursive doubling 1
3. $0 \leftrightarrow 2, 1 \leftrightarrow 3, 4 \leftrightarrow 6, 5 \leftrightarrow 7$	Recursive doubling 2
4. $0 \leftrightarrow 4, 1 \leftrightarrow 5, 2 \leftrightarrow 6, 3 \leftrightarrow 7$	Recursive doubling 3
5. $0 \leftarrow 2, 1 \leftarrow 3, 2 \leftarrow 4, 3 \leftarrow 5$ $4 \leftarrow 6, 5 \leftarrow 7, 6 \leftarrow 0, 7 \leftarrow 1,$	Bruck 2
6. $1 \leftarrow 0, 3 \leftarrow 2, 5 \leftarrow 4, 7 \leftarrow 6$	Binomial-tree 3

Table 4.1: Communication patterns.

affinity causes adjacent ranks to be two apart in the sequence. This leads to balanced communication in both directions with the Ring pattern.

4.4.1 Communication patterns

We considered the communication patterns specified in table 4.1. We also considered a few communication patterns that have multiple phases. In each phase, they use one of the simpler patterns mentioned in table 4.1. They synchronize with a barrier after all the phases are complete. These patterns typically arise in the following collective communication calls: (i) binomial-tree based broadcast, (ii) binomial-tree based scatter, (iii) Bruck algorithm for all-gather, and (iv) recursive doubling based all-gather.

4.4.2 Experimental Results

Figures 4.8 - 4.13 show the performance of the different affinity schemes with the communication patterns mentioned in table 4.1. *In these results, we report the performance of the message with the lowest bandwidth, because this will typically be the bottleneck for an application.*

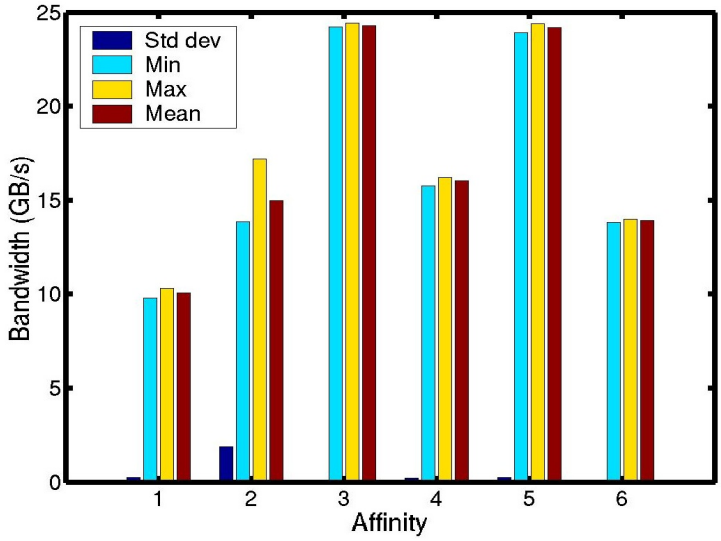

Figure 4.8: Performance of the following affinities: (1) Overlap, (2) Default, (3) EvenOdd, (4) Identity, (5) Leap2, (6) Ring – Ring pattern.

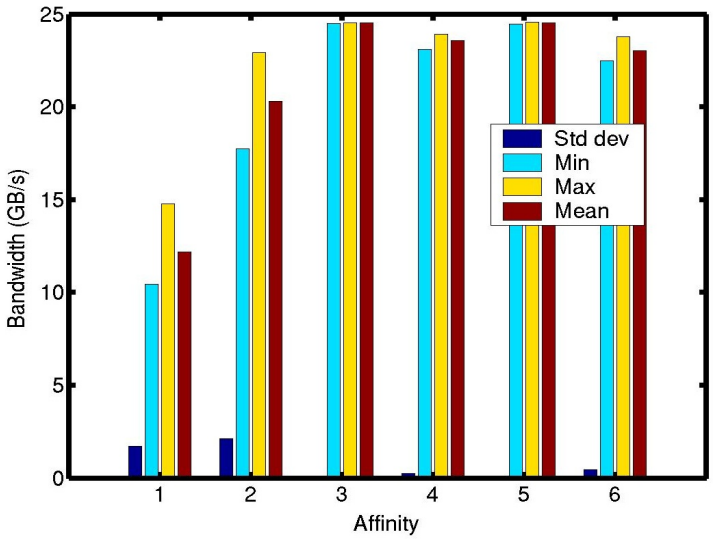

Figure 4.9: Performance of the following affinities: (1) Overlap, (2) Default, (3) EvenOdd, (4) Identity, (5) Leap2, (6) Ring – first phase of Recursive Doubling.

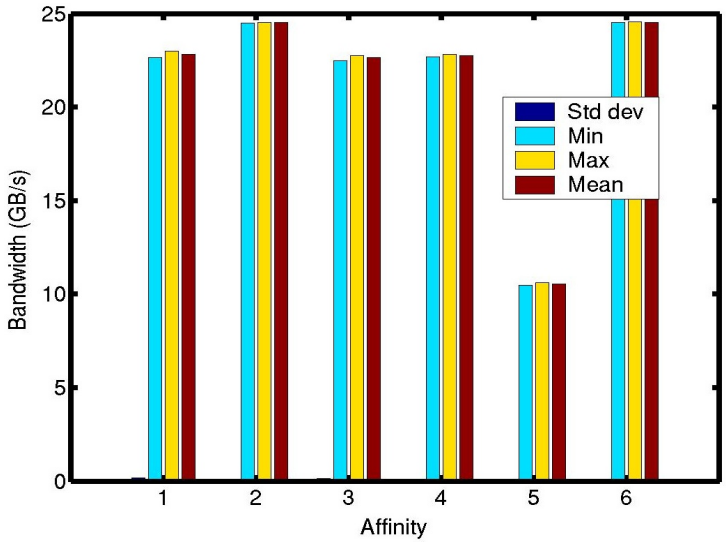

Figure 4.10: Performance of the following affinities: (1) Overlap, (2) Default, (3) EvenOdd, (4) Identity, (5) Leap2, (6) Ring – second phase of Recursive Doubling.

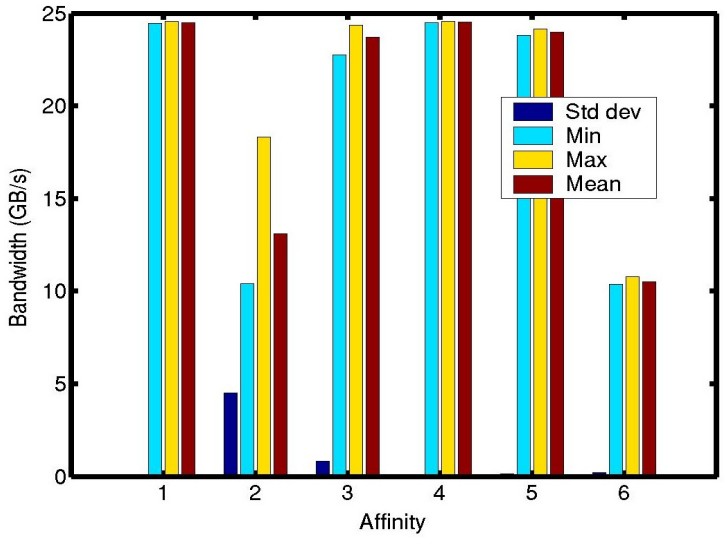

Figure 4.11: Performance of the following affinities: (1) Overlap, (2) Default, (3) EvenOdd, (4) Identity, (5) Leap2, (6) Ring – third phase of Recursive Doubling.

Figure 4.8 shows results with the Ring communication pattern. We can see that the Overlap affinity gets less than half the performance of the best patterns. This is not surprising, because this affinity was developed to give a lower bound on the performance on the ring pattern, with a large number of overlaps. The Ring affinity does not have any overlap, but it has a very unbalanced load, with all the transfers going counter-clockwise, leading to poor performance. The Leap2 pattern does not have any overlapping paths in each direction, and so it gives good performance. The Identity affinity has only one explicit overlap in each direction, which by itself does not degrade performance. However, it also has a few paths that go half-way across the ring, and these cause additional overlap whichever way they go, leading to loss in performance. The EvenOdd affinity has a somewhat similar property; however, the paths that go half-way across have a direction in which they do not cause an overlap. It appears that these good paths are taken, and so the performance is good.

The default affinity is somewhat random. So, we shall not explicitly give reasons for its performance below. We show results with the default affinity primarily to demonstrate the improvement in performance that can be obtained by explicitly specifying the affinity. The randomness in this affinity also leads to a higher standard deviation than for the other affinities.

Figure 4.9 shows results with the first phase of Recursive Doubling. The Overlap pattern has all paths going half-way across. So, there is extensive overlap, and consequently poor performance. The other four deterministic patterns yield good performance. The Ring affinity has no overlap in each direction. The other three have overlaps; however, the

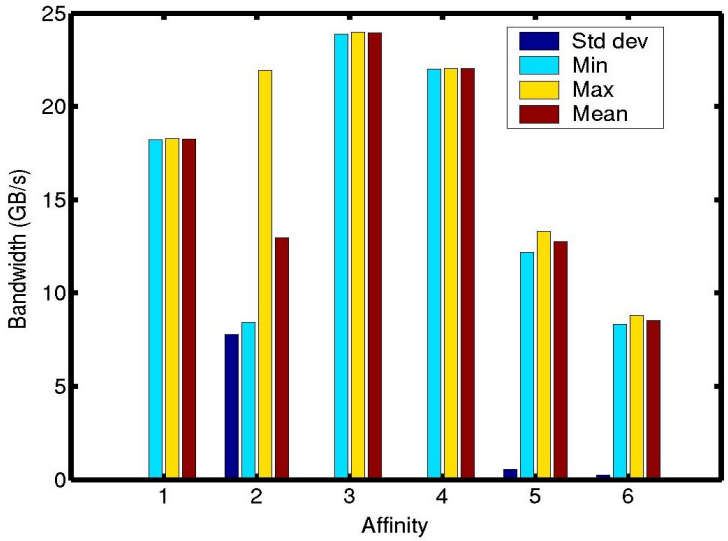

Figure 4.12: Performance of the following affinities: (1) Overlap, (2) Default, (3) EvenOdd, (4) Identity, (5) Leap2, (6) Ring – second phase of Bruck algorithm for all-gather.

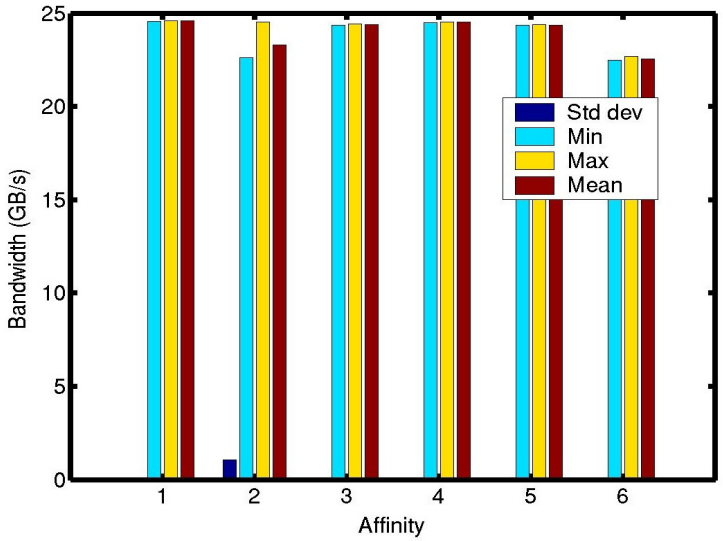

Figure 4.13: Performance of the following affinities: (1) Overlap, (2) Default, (3) EvenOdd, (4) Identity, (5) Leap2, (6) Ring – third phase of Binomial Tree.

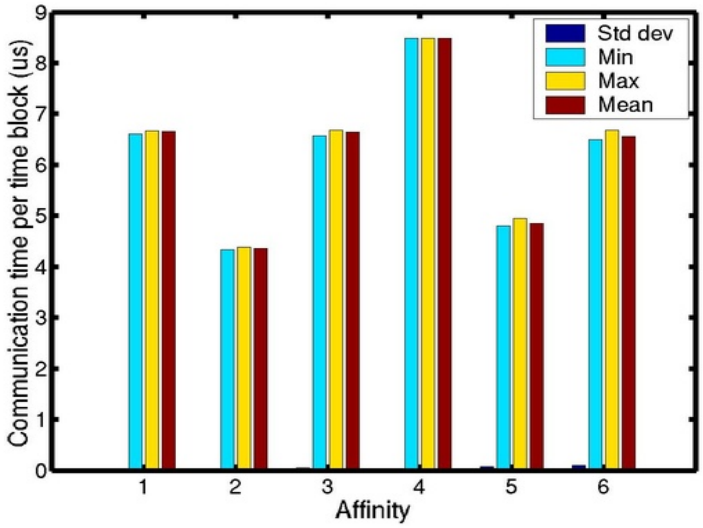

Figure 4.14: Performance of the following affinities: (1) Overlap, (2) Default, (3) EvenOdd, (4) Identity, (5) Leap2, (6) Ring, for the Particle transport application : communication time.

transfers can be assigned to distinct rings on the EIB such that there is no overlap in each ring, leading to good performance.

Figure 4.10 shows results with the second phase of Recursive Doubling. Leap2 alone has poor performance, because it has all paths going half-way across. The Ring affinity has overlaps that can be placed on different rings. The other deterministic patterns do not have any overlap in the same direction. The third phase of recursive doubling is shown in figure 4.11. The Ring affinity alone has poor performance, among the deterministic affinities, because it has all paths going half-way across. The other deterministic patterns have overlaps that can be placed on distinct rings, leading to good performance.

Figure 4.12 shows results for the second phase of the Bruck algorithm[1]. The EvenOdd affinity performs best, because it does not

[1]The 1st phase of Bruck algorithm is identical to ring pattern, and the 3rd phase is identical to the 3rd phase of Recursive Doubling.

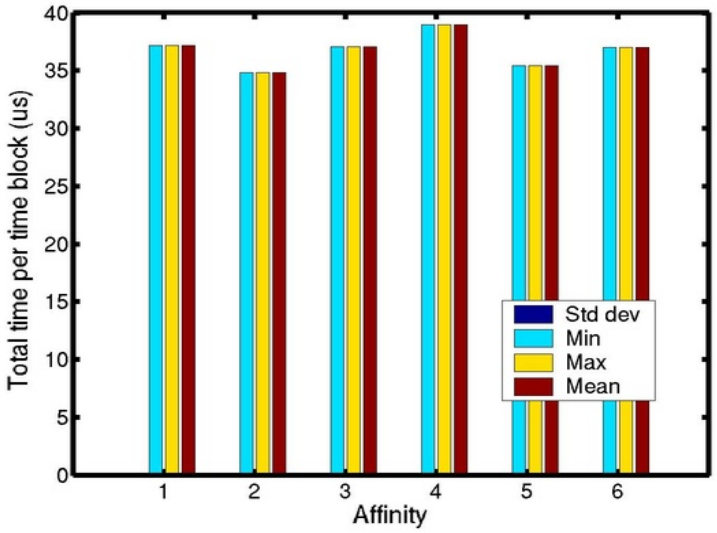

Figure 4.15: Performance of the following affinities: (1) Overlap, (2) Default, (3) EvenOdd, (4) Identity, (5) Leap2, (6) Ring, for the Particle transport application : total time.

have any overlap in the same direction. Identity too does not have any overlap, and gives good performance. However, its performance is a little below that of EvenOdd. The Ring affinity has all transfers going in the same direction and gets poor performance. The Overlap affinity has no overlap, but has unbalanced load, with the counter-clockwise ring handling six of the eight messages, which reduces its performance. The Leap2 affinity has several transfers going half-way across, which result in overlap with either choice of direction, and consequently lead to poor performance.

All affinities perform well on the third phase of binomial-tree[2], shown in figure 4.13. In the Ring affinity, all messages go in the same direction. However, since there are only four messages in total, the performance is still good. All the other affinities have two transfers in each direction,

[2]The first phase of binomial-tree has only one message, and the second phase has only two messages. Therefore, there is no contention in these phases.

and each of these can be placed on a distinct ring, yielding good performance.

We next consider the performance of the above affinity schemes in a practical application. This is a Monte Carlo application for particle transport, which tracks a number of random walkers on each SPE [43]. Each random walker takes a number of steps. A random walker may be terminated on occasion, causing a load imbalance. In order to keep the load balanced, we use the diffusion scheme, in which a certain fraction of the random walkers on each SPE is transferred to a neighbor after each step. SPE i considers SPEs $i-1$ (modulo N) and $i+1$ (modulo N) as its neighbors when we use N SPEs. We can see a factor of two difference between the communication costs for the best and worst affinities in figure 4.14. Figure 4.15 shows a difference in total application performance of over 10% between the best and worst affinities.

The above results for different communication patterns showed a significant difference in the performance across different affinities. This indicated to the need for application communication pattern based mapping of threads to SPEs for obtaining good performance.

4.5 Affinity on a Cell Blade

Communication on a Cell Blade is asymmetric, with around 30 GB/s theoretically possible from Cell 0 to Cell 1, and around 20 GB/s from Cell 1 to Cell 0. However, as shown in table 4.2, communication between a single pair of SPEs on different processors of a Blade yields bandwidth much below this theoretical limit[3]. In fact, this limit is not reached even when multiple SPEs communicate, for messages of size up to 64 KB each.

[3]This result is consistent with those presented in [40] for the QS21 Blade.

Processor 1 to 0	Processor 0 to 1	Bandwidth 1 to 0	Bandwidth 0 to 1
1	0	3.8 GB/s	
0	1		5.9 GB/s
1	1	3.8 GB/s	5.9 GB/s
2	2	3.4 GB/s	5.0 GB/s
3	3	2.8 GB/s	3.6 GB/s
4	4	2.6 GB/s	2.7 GB/s
5	5	2.0 GB/s	2.0 GB/s
6	6	1.6 GB/s	1.6 GB/s
7	7	1.3 GB/s	1.3 GB/s
8	8	1.2 GB/s	1.2 GB/s

Table 4.2: Performance of each message for different numbers of SPEs communicating simultaneously across processors on a Cell Blade with 64 KB messages each.

Affinity	Bandwidth
Identity	3.8 GB/s
EvenOdd	3.5 GB/s
Ring	3.9 GB/s
Worst case	1.3 GB/s

Table 4.3: Performance for different affinities on a Cell Blade (16 SPEs) for 64 KB messages with the Ring communication pattern.

As shown above, the bandwidth attained by messages between SPEs on different processors is much lower than that between SPEs on the same processor. So, these messages become the bottleneck in the communication. In this case, the affinity within each SPE is not as important as the partitioning of threads amongst the two processors. We have created a software tool that evaluates each possible partitioning of the threads amongst the two processors and chooses the one with the smallest communication volume. This tool takes into account the asymmetry in the bandwidth between the two processors, and so a partition sending more data will be placed on processor 0. Table 4.3 shows that such partitioning is more important than the affinity within each processor. In that figure, the software mentioned above was used to partition the threads for a ring communication pattern, and certain affinities studied above were used within each processor. One case, however, considered a worst-case partitioning where each transfer is between a pair of SPE on different processors.

4.6 Communication Model

Based on the understanding from the above mentioned experimental analysis, we have developed a tool with an aim to find the optimal mapping. The main purpose of creating a quantitative model is that we can find an optimal affinity for an arbitrary communication pattern, without being ingenious. The model evaluate all possible affinities other than the symmetrically similar mappings, and chooses the best among them. Hence, the model has a computational complexity of (n-1)!/2. The model was developed based on the following guiding principles:

- For good performance, the communication load should be distributed equally across all the four EIB rings.
- Each ring can simultaneously support three data transfers, provided that the paths for these transfers do not overlap. Model should look for a mapping, where the paths taken by the messages do not overlap often.
- The model takes into account the asymmetry in the bandwidth between two processors, and so a partition sending more data will be placed on processor 0.

4.6.1 Evaluation of the model

We now evaluate the effectiveness of the model by using it for determining affinities for some standard communication patterns and also for the real application. Model's affinity in the figures denotes the affinity determined by using the tool. Figure 4.16 shows the effective bandwidths achieved for the Ring communication pattern using different mappings. Figure 4.17 shows the effective bandwidths for the first phase of Recursive Doubling. These results show that the performance of mapping given by the model is not as good as the mapping which attains the maximum possible bandwidth, however it is better than the default mapping.

We next consider the performance of the communication model on the Monte Carlo application. We can observe from the figures 4.18 and 4.19 that the affinity given by the model results in sub-optimal performance. This could be due to the fact that the three guiding principles used in building the model are not sufficient to

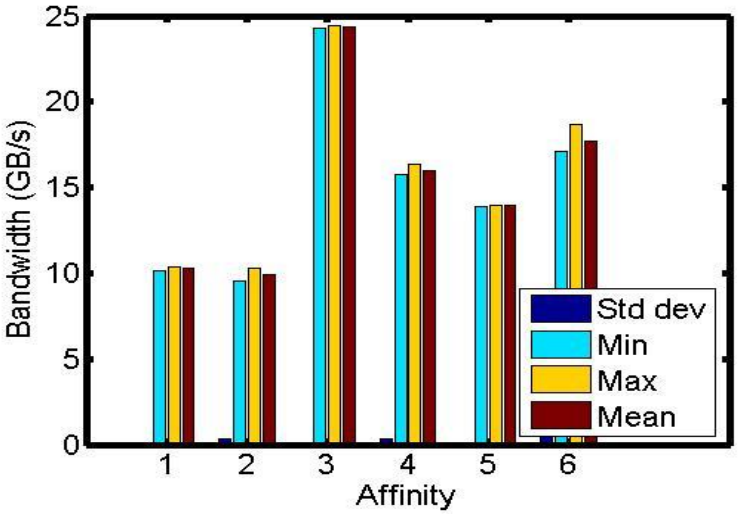

Figure 4.16: Performance with the following mappings: 1) Overlap 2) Default 3) EvenOdd 4) Identity 5) Ring 6) Model's affinity.

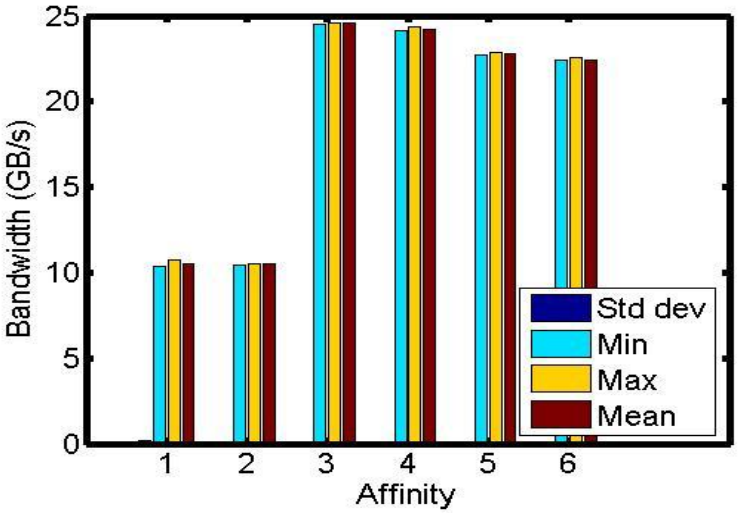

Figure 4.17: Performance with the following mappings: 1) Overlap 2) Default 3) EvenOdd 4) Identity 5) Ring 6) Model's affinity.

accurately represent the communication network (EIB) operation. For example, we observed a poor bandwidth performance even with three non overlapping messages between SPEs going in the same direction.

This occurs when at least two of them are of path length two or more, and go across the sides of the ring (i.e., through PPE-MIC or BIE-IOIF1 in figure 4.1). This non intuitive behavior of the non overlapping messages is not thoroughly included in the model. The traffic to the main memory, which also goes through the same EIB significantly affecting the inter-SPE communication, was also not considered in the model.

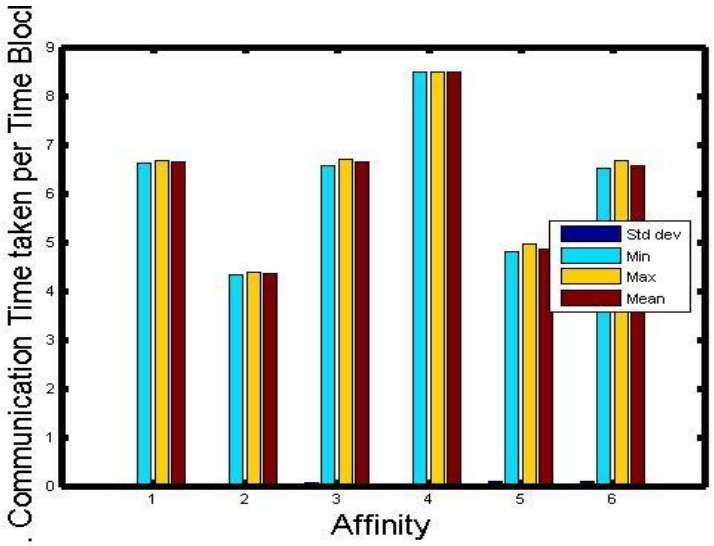

Figure 4.18: Performance with the following mappings: 1) Overlap 2) Default 3) EvenOdd 4) Identity 5) Ring 6) Model's affinity.

Another issue is that in all the experiments, we considered only messages with equal size. We observed that messages with unequal sizes results in different behavior in some cases. For example, we observed that the above noted peculiar behavior of non overlapping messages does not occur if the messages are of different sizes. We also assumed symmetry in rotating the affinity, to reduce the number of affinities tested from 8! to 7!, however, our experiments with some communication patterns indicated that such symmetry does not exist. We could modify our model by taking into consideration all these aspects

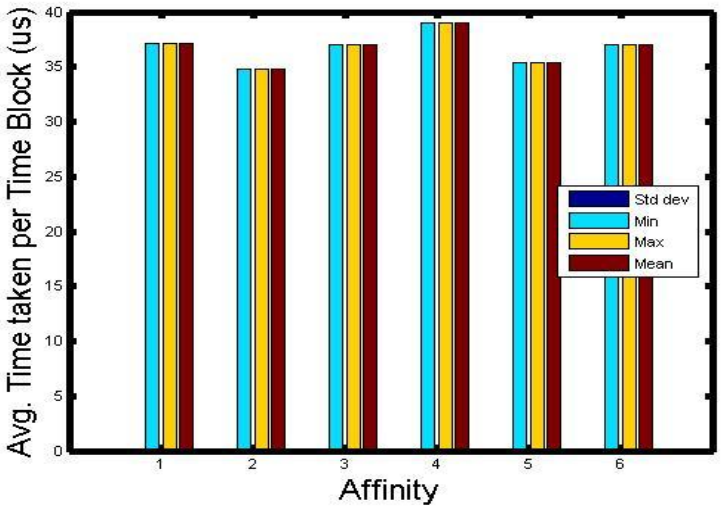

Figure 4.19: Performance with the following mappings: 1) Overlap 2) Default 3) EvenOdd 4) Identity 5) Ring 6) Model's affinity.

to make it more complete and robust.

5. Application Specific Topology Aware Mapping: A Load Balancing Application

We wish to show that suitable assignment of tasks to cores of a massively parallel machine can reduce the communication overhead significantly. We consider a new load balancing algorithm that we have developed to demonstrate this. We first explain its computation and communication pattern and then describe how a topology aware mapping reduces its communication costs.

5.1 Introduction to the Application

Quantum Monte Carlo (QMC) is a class of quantum mechanics-based methods for electronic structure calculations [45, 46, 47]. These methods can achieve much higher accuracy than well-established techniques such as Density Functional Theory (DFT) [48] by directly treating the quantum-mechanical many-body problem. However, this increase in accuracy comes at the cost of substantially increased computational effort. The most common QMC methods are nominally cubic scaling,

but have a large prefactor, making them several orders of magnitude more costly than the less-accurate DFT calculation. On the other hand, QMC can today effectively use the largest parallel machines, with $O(10^5)$ processing elements, while DFT can not use these machines routinely. However, with the expected arrival of machines with orders of magnitude more processing elements than common today, it is important that all the key algorithms of QMC are optimal and remain efficient at scale.

Diffusion Monte Carlo (DMC) is the most popular modern QMC technique for accurate predictions of materials and chemical properties at zero temperature. It is implemented in software packages such as CASINO [49], CHAMP [50], QMCPack [51], and QWalk. Unlike some Monte Carlo approaches, this method is not trivially parallel and requires communications throughout the computation.

The DMC computation involves a set of random walkers, where each walker represents a quantum state. At each time step, each walker moves to a different point in the configuration space, with this move having a random component. Depending on the energy of the walker in this new state relative to the average energy of the set of walkers (or a reference energy related to the average), the walker might be either terminated or new walkers are created at the same position. Alternatively, weights may be associated with each walker, and the weights are increased or decreased appropriately. Over time, this process creates a load imbalance, and the set of walkers (and any weights) must be re-balanced. For optimum statistical efficiency, this re-balancing should occur every single move [52].

DMC is parallelized by distributing the set of walkers over the available compute cores. The relative cost of load balancing the walkers

as well as the inefficiency from the statistical fluctuations in walker count can be minimized if the number of walkers per compute element is kept large. This approach has typically been used on machines with thousands to tens of thousands of cores. However, with increasing core count the total population of walkers is increased. This is undesirable since (1) there is an equilibration time for each walker that does not contribute to the final statistics and physical result, (2) it is usually preferable to simulate walkers for longer times, enabling any long term trends or correlations to be determined, (3) the amount of memory per compute element is likely to reduce in future, necessitating smaller populations per compute element. On the highest-end machines, it is desirable to use very few walkers (one or two) per compute element and assign weights to the walkers, instead of adding or subtracting walkers, to avoid excessively large walker counts and to avoid the large fluctuations in computational effort that would result from even minor fluctuations in walker count.

Since load balancing is in principle a synchronous, blocking operation, requiring communication between all compute elements, it is important that the load balancing method is highly time efficient and makes very effective use of the communications network, minimizing the number and size of messages that must be sent. It is also desirable that the algorithm is simple to enable optimization of the messaging for particular networks, and to simplify use of latency hiding techniques through overlap of computation and communications. We note that CASINO [49] recently transitioned [53] to using asynchronous communications and suspect that other codes may use some of these techniques, but apart from [53], they have not been formally described.

In this chapter we first discuss our new load balancing algorithm which can be used to load balance computations involving near identical independent tasks such as those in DMC (we consider each random walker a task in the description of our load balancing algorithm). The algorithm has the interesting feature that each process needs to receive tasks from at most one other process. We optimize this algorithm on the peta-flop Jaguar supercomputer and show, using data from the simulation of the Cr_2 molecule, that it improves performance over the existing load balancing implementation of the QWalk code by up to 30% on 120,000 cores. Moreover, due to the optimal nature of the algorithm we expect its utility and effectiveness to increase with the multiple orders of magnitude increase in compute elements expected in the coming years.

5.1.1 Load Balancing Model Definitions

Dynamic load balancing methods often consist of the following three steps. (i) In the *flow computation* step, we determine the number of tasks that need to be sent by each process to other processes. (ii) In the *task identification* step, we identify the actual tasks that need to be sent by each process. (iii) In the *migration* step, the tasks are finally sent to the desired processes. Since we deal with identical independent tasks, the second step is not important; any set of tasks can be chosen. Our algorithm determines the flow (step (i)) such that step (iii) will be efficient, under certain performance metrics.

We assume that a collection of P processes need to handle a set of T identical tasks (that is, each task requires the same computation time), which can be executed independently. Before the load balancing

phase, the number of tasks with process i, $1 \leq i \leq P$, is T_i. After load balancing, each process will have at most $\lceil T/P \rceil$ tasks (we are assuming that the processors are homogeneous, and therefore process tasks at the same speed). This redistribution of tasks is accomplished by having each process i send t_{ij} tasks to processes j, $1 \leq i, j \leq P$, where non-zero values of t_{ij} are determined by our algorithm for flow computation, which we describe in section 5.3. Of course, most of the t_{ij}s should be zero, in order to reduce the total number of messages sent. In fact, at most $P-1$ of the possible $P(P-1)$ values of t_{ij} will be non-zero in our algorithm.

The determination of t_{ij}s is made as follows. The processes perform an "all-gather" operation to collect the number of tasks on each process. Each process k independently implicitly computes the flow (all non-zero values of t_{ij}, $1 \leq i, j \leq P$) using the algorithm in section 5.3, and then explicitly determines which values of t_{kj} and t_{jk} are non-zero, $1 \leq j \leq P$.

The algorithm to determine non-zero t_{ij}s takes $O(P)$ time, and is fast in practice. We wish to minimize the time taken in the actual migration step, which is performed in a decentralized manner by each process. In some load balancing algorithms [54], a process may not have all the data that it needs to send, and so the migration step has to take place iteratively, with a process sending only data that it has in each iteration. In contrast, the t_{ij}s generated by our algorithm never require sending more data than a process initially has, and so the migration step can be completed in one iteration. In fact, no process *receives* a message from more than one process, though processes may need to *send* data to multiple processes.

The outline of the rest of the chapter is as follows. We summarize related work in section 5.2. In section 5.3, we describe our algorithm

for dynamic load balancing. We first describe the algorithm when T is a multiple of P, which is the ideal case, and then show how the algorithm can be modified to deal with the situation when T is not a multiple of P. We then report results of empirical evaluation of our method and comparisons with an existing QMC dynamic load balancing implementation, in section 5.4. We finally summarize this chapter in section 5.5.

5.2 Related Work

Load balancing has been, and continues to be, an important research issue. Static partitioning techniques try to assign tasks to processes such that the load is balanced, while minimizing the communication cost. This problem is NP-hard in most reasonable models, and thus heuristics are used. Geometric partitioning techniques can be used when the tasks have coordinate information, which provide a measure of distance between tasks. Graph based models abstract tasks as weighted vertices of a graph, with weights representing computational loads associated with tasks. Edges between vertices represent communication required, when one task needs information on another task. A variety of partitioning techniques have been studied, with popular ones being spectral partitioning [55, 56] and multi-level techniques [57, 58, 59, 60], and have been available for a while in software such as Chaco and Metis.

Dynamic load balancing schemes start with an existing partition, and migrate tasks to keep the load balanced, while trying to minimize the communication cost of the main computation. A task is typically sent to a process that contains neighbors of the task in the communication graph, so that the communication cost of the main computation is

minimized[1]. Other schemes make larger changes to the partitions, but remap the computation such that the cost of migration is small [61].

The diffusion scheme is a simple and well-known scheme sending data to neighboring processes [62, 63, 64, 65]. Another scheme, proposed in [54], is also based on sending tasks to neighbors. It is based on solving a linear system involving the Laplacian of the communication graph. Both these schemes require the tasks to be arbitrarily divisible for the load balancing to work. For example, one should be able to send 0.5 tasks, 0.1 tasks, etc. Modified versions of diffusive type schemes have also been proposed which remove restrictions on arbitrary divisibility [66]. Multi-level graph partitioning based dynamic schemes are also popular [69]. Hyper-graphs generalize graphs using hyper-edges, which are sets of vertices with cardinality not limited to two. Hyper-graph based partitioning has also been developed [67]. Software tools, such as, JOSTLE [68], ParMetis, and Zoltan are available, implementing a variety of algorithms.

There has been much work performed on load balancing independent tasks (bag of tasks) in the distributed and heterogeneous computing fields [70, 71, 72, 73]. Many of the scheduling algorithms try to minimize the makespan, which can be considered a type of load balancing. They consider issues such as differing computing power of machines, online scheduling, etc.

Within the context of QMC and DMC, we are not aware of any published work specifically focusing on the algorithms used for load balancing, although optimizations to existing implementations have been described [53]. Since all QMC codes must perform a load

[1]When tasks are fairly independent, as in QMC, it is reasonable to model it as a complete graph, indicating that a task can be migrated to any process.

balancing step, each must have a reasonably efficient load balancing implementation, at least for modest numbers of compute elements. However, the methods used have not been formally described and we do not believe any existing methods share the optimality features of the algorithm described below.

5.3 The Alias Method Based Algorithm for Dynamic Load Balancing

Our algorithm is motivated by the alias method for generating samples from discrete random distributions. We therefore refer to our algorithm as the Alias method for dynamic load balancing. There is no randomness in our algorithm. It is, rather, based on the following observation used in a deterministic pre-processing step of the alias method for the generation of discrete random variables. If we have P bins containing kP objects in total, then it is possible to re-distribute the objects so that each bin receives objects from at most one other bin, and the number of objects in each bin, after the redistribution, is exactly k. Walker [75] showed how this can be accomplished in $O(P \log P)$ time. This time was reduced to $O(P)$ by [74] using auxiliary arrays. In Algorithm 1 below, we describe our in-place implementation that does not use auxiliary arrays, except for storing a permutation vector.

We assume that the input to Algorithm 1 is an integer array A containing the number of objects in each bin. Given A, we can compute k easily in $O(P)$ time, and will also partition it around k in $O(P)$ time so that all entries with $A[i] < k$ occur before any entry with $A[j] > k$. We will assume that $A[i] \neq k$, because other bins do not need to be considered

– they have the correct number of elements already, and our algorithm does not require redistribution of objects to or from a bin that has k objects. If we store the permutation while performing the partitioning, then the actual bin numbers can easily be recovered after Algorithm 1 is completed. This algorithm runs only with $P \geq 2$, because otherwise all the bins already have k elements each. We assume that a pre-processing step has already accomplished the above requirements in $O(P)$ time.

Algorithm 1:

Input: An array of non-negative integers $A[1 \cdots P]$ and an integer $k > 0$, such that $\sum_{i=1}^{P} A[i] = kP$, entries of A have been partitioned around k, and $P \geq 2$. $A[i]$ gives the number of objects in bin i, and $A[i] \neq k$.

Output: Arrays $S[1 \cdots P]$ and $W[1 \cdots P]$, where $S[i]$ gives the bin from which bin i should get $W[i]$ objects, if $S[i] \neq 0$.

Algorithm:

1. Initialize arrays S and W to all zeros.
2. $s \leftarrow 1$.
3. $l \leftarrow \min\{j | A[j] > k\}$.
4. while $l > s$
   (a) $S[s] \leftarrow l$.
   (b) $W[s] \leftarrow k - A[s]$.
   (c) $A[l] \leftarrow A[l] - W[s]$.
   (d) if $A[l] < k$ then
      i. $l \leftarrow l + 1$.

(e) $s \leftarrow s + 1$.

It is straightforward to see the correctness of Algorithm 1 based on the following loop invariants at the beginning of each iteration in step 4: (i) $A[i] \geq k,\ l \leq i \leq P$, (ii) $0 \leq A[i] < k,\ s \leq i \leq l - 1$, and (iii) $A[i] + W[i] = k,\ 1 \leq i \leq s - 1$. Since bin l needs to provide at most k objects to bin s, it has a sufficient number of objects available, and also as a consequence of the same fact, $A[l]$ will not become negative after giving $W[s]$ objects to bin s. The last clause of the loop invariant proves that all the bins will have k objects after the redistribution. We do not formally prove the loop invariants, since they are straightforward.

In order to evaluate the time complexity, note that in the while loop in step 4, l and s can never exceed P. Furthermore, each iteration of the loop takes constant time, and s is incremented once each iteration. Therefore, the time complexity of the while loop is $O(P)$. Step 3 can easily be accomplished in $O(P)$ time. Therefore the time complexity of this algorithm is $O(P)$.

Load balancing when T is a multiple of P: Using Algorithm 1, a process can compute t_{ij}s as follows, if we associate each bin with a process[2] and the number of objects with the number of tasks:

$$t_{S[i]i} \leftarrow W[i],\ S[i] \neq 0. \tag{5.1}$$

All other t_{ij}s are zero. Of course, one needs to apply the permutation obtained from the partitioning before performing this assignment. Note

[2]In our algorithm, processes that already have a balanced load do not participate in the redistribution of tasks to balance the load. Therefore, we use P to denote the number of processes with *unbalanced loads* in the remainder of the theoretical analysis.

that the loop invariant mentioned for Algorithm 1 also shows that a process always has sufficient data to send to those that it needs to; it need not wait to receive data from any other process in order to have sufficient data to send, unlike some other dynamic load balancing algorithms [54].

Load balancing when T is not necessarily a multiple of P: The above case considers the situation when the total number of tasks is a multiple of the total number of processes. We can also handle the situation when this is not true, using the following modification. If there are T tasks and P processes, then let $k = \lceil T/P \rceil$. For balanced load, no process should have more than k tasks. We modify the earlier scheme by adding $kP - T$ fake "phantom" tasks. This can be performed conceptually by incrementing $A[i]$ by one for $kP-T$ processes before running Algorithm 1 (and even before the pre-processing steps involving removing of entries with $A[i] = k$ and partitioning). The total number of tasks, including the phantom ones, is now kP, which is a multiple of P. So Algorithm 1 can be used on this, yielding k tasks per process. Some of these are phantom tasks, and so the number of tasks is at most k, rather than exactly k. We can account for the phantom tasks by modifying the array S as follows, after completion of Algorithm 1. Let F be the set of processes to which the fake phantom tasks were added initially (by incrementing their A entry). For each $j \in F$, define $r_j = \min\{i | S[i] = j\}$. If r_j exists, then set $W[r_j] \leftarrow W[r_j] - 1$. This is conceptually equivalent to making each process that initially had a phantom task to send this task to the first process to whom it sends anything. Note that on completion of the algorithm, no process has more than two phantom tasks, because in the worst case, it had one initially, and then received one more. So the total

number of tasks on any process after redistribution will vary between $k-2$ to k. The load is still balanced, because we only require that the maximum load not exceed $\lceil T/P \rceil$ after the redistribution phase[3]. This modified algorithm can be implemented with the same time complexity as the original algorithm.

5.4 Empirical Results

5.4.1 Experimental Setup

The experimental platform is the Cray XT5 Jaguar supercomputer at ORNL. In running the experiments, we have two options regarding the number of processes per node. We can either run one process per node or one process per core. QMC software packages were originally designed to run one MPI process per core. The trend now is toward one MPI process per node, with OpenMP threads handling separate random walkers on each core. Qmcpack already has this hybrid parallelization implemented, and some of the other packages are expected to have it implemented in the near future. We assume such a hybrid parallelization, and have one MPI process per node involved in the load balancing step.

In our experiments, we consider a granularity of 24 random walkers per node, that is, 2 per core. This is a level of granularity that we desire for QMC computations in the near future. Such scalability is currently limited by the periodic collective communication and load balancing that is required.

[3] In our implementation, the phantom tasks are not actually sent, and they do not even exist in memory.

Both these are related in the following manner. The first step leads to termination or creation of new walkers, which in turn requires load balancing. There is some flexibility in the creation and termination of random walkers. Ideally, the load balancing results both in reduced wall clock time per step (of all walkers) and an improved statistical efficiency.

We note that there is some flexibility in the creation and termination of random walkers. Ideally the load balancing is performed after every time step, to obtain the best statistical error and to minimize systematic errors due to the finite sized walker population. However, the overhead on large parallel machines can hinder this, and so one may perform them every few iterations instead. Our goal is to reduce these overheads so that these steps can be performed after every time step. For large physical systems where the computational cost per step is very high, these overheads may be relatively small compared with the computation cost. However, for small to moderate sized physical systems, these overheads can be large, and we wish to efficiently apply QMC even to small physical systems on the largest parallel computational systems.

We consider a small system, a Cr_2 molecule, with an accurate multideterminant trial wavefunction. The use of multideterminants increases computational time over the use of a single determinant. However, it provides greater accuracy, which we desire when performing a large run. The computation time per time step per walker is then around 0.1 seconds. The two collective steps mentioned above consume less than 10% of the total time on a large machine (the first step does not involve just collective communication, but also involves other global decisions, such as branching). Even then, on 100,000 cores, this is equivalent to wasting 10,000 cores. We can expect these collective steps

to consume a larger fraction of time at even greater scale.

In evaluating our load balancing algorithm, we used samples from the load distribution observed in a long run of the above physical system. Depending on the details of the calculations, the amount of data to be transferred for each random walker can vary from 672B to 32KB for Cr_2. We compared our algorithm against the load balancing implementation in QWalk. The algorithm used in QWalk is optimal in the maximum number of tasks sent by any processor and in the total number of tasks sent by any processor, but not on the maximum or total number of messages sent; these are bounded by the maximum imbalance and the sum of load imbalances respectively. One may, therefore, expect that algorithm to be more efficient than ours for a sufficiently large task sizes, and ours to be better for small sizes. Also, the time taken for the flow computation in QWalk is O(P + total_load), where P is the number of nodes.

Each experiment involved 11 runs. As we show later, inter-job contention on the network can affect the performance. In order to reduce its impact, we ignore the results of the run with the largest total time. In order to avoid bias in the result, we also drop the result of the run with the smallest time. For a given number of nodes, all runs for all task sizes for both algorithms are run on the same set of nodes, with one exception mentioned later.

5.4.2 Results

Our testing showed that the time taken for the alias method is linear in the number of nodes, as expected theoretically (not shown). The maximum time taken by any node can be considered a measure of the

performance of the algorithm, because the slowest processor limits the performance. Figure 5.1 shows the average, over all the runs, of the maximum time for the following components of the algorithm. (We refer to it in the figure caption as the 'basic alias method', in order to differentiate it from a more optimized implementation described later.) Note that the maximum for each component may occur on different cores, and so the maximum total time for the algorithm over all the cores may be less than the sum of the maximum times of each component. We can see that communication operations consume much of the time, and the flow computation is not the dominant factor, even with a large number of nodes. The MPI_Isend and MPI_Irecv operations take little time. However MPI_Waitall and MPI_Allgather consume a large fraction of the time. It may be possible to overlap computation with communication to reduce the wait time. However, the all-gather time is still a large fraction of the total time.

In interpreting the plot in figure 5.1, one needs to note that it is drawn on a semi-log scale. The increase in time, which appears exponential with the number of cores, is not really so. A linear relationship would appear exponential on a semi-log scale. On the other hand, one would really expect a sub-linear relationship for the communication cost. The all-gather would increase sub-linearly under common communication cost models. In the absence of contention, the cost of data transfer need not increase with the number of cores for the problem considered here; the maximum imbalance is 4, each node has 6 communication links, and so, in principle, if the processes are ideally ordered, then it is possible for data to travel on different links to nearby neighbors which would be in need of tasks. The communication time can, thus, be held constant. We

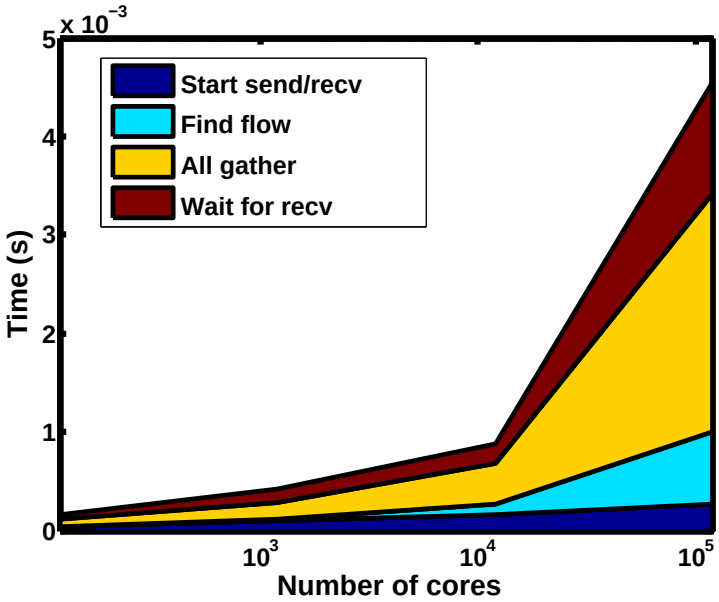

Figure 5.1: Maximum time taken for different components of the basic Alias method with task size 8KB.

can see from this figure that the communication cost (essentially the wait time) does increase significantly. The communication time for 12,000 cores is 2-3 times the time without contention, and the time with 120,000 cores is 4-6 times that without contention. The cause for contention is that the routing on this machine uses fixed paths between pairs of nodes, and sends data along the x coordinate of the torus, in the direction of the shortest distance, then in the y direction, and finally in the z direction[4]. Multiple messages may need to share a link, which causes contention.

In fig. 5.2, we consider the mean value of the different components in each run, and plot the average of this over all runs. We can see that the wait time is very small. The reason for this is that many of the nodes have balanced loads. The limiting factor for the load balancing algorithm is the few nodes with large work.

[4]Personal communication from James Buchanan, OLCF, ORNL.

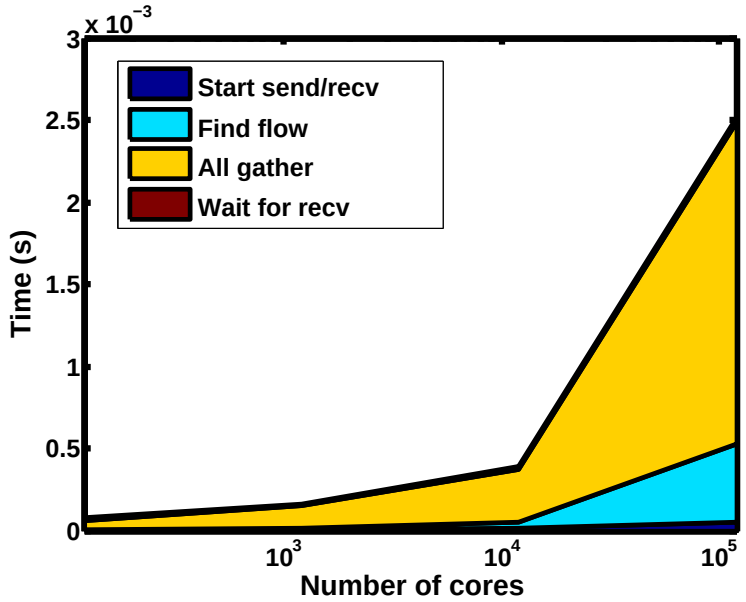

Figure 5.2: Mean time taken for different components of the basic Alias method with task size 8KB.

We next optimize the alias method to reduce contention. We would like nodes to send data to nearby nodes. We used a heuristic to accomplish this. We obtained the mapping of node IDs to x, y, and z coordinates on the 3-D torus. We also found a space-filling Hilbert curve that traverses these nodes. (A space-filling curve tries to order nodes so that nearby nodes are close by on the curve.) At run time, we obtain the node IDs, and create a new communicator that ranks the nodes according to their relative position on the space-filling curve. We next changed the partitioning algorithm so that it preserves the order of the space-filling curve in each partition. We also made slight changes to the alias algorithm so that it tries to match nodes based on their order in the space filling curve. The creation of a new communicator is performed only once, and the last two steps don't have any significant impact on the time taken by the alias method. Thus, the improved algorithm is no

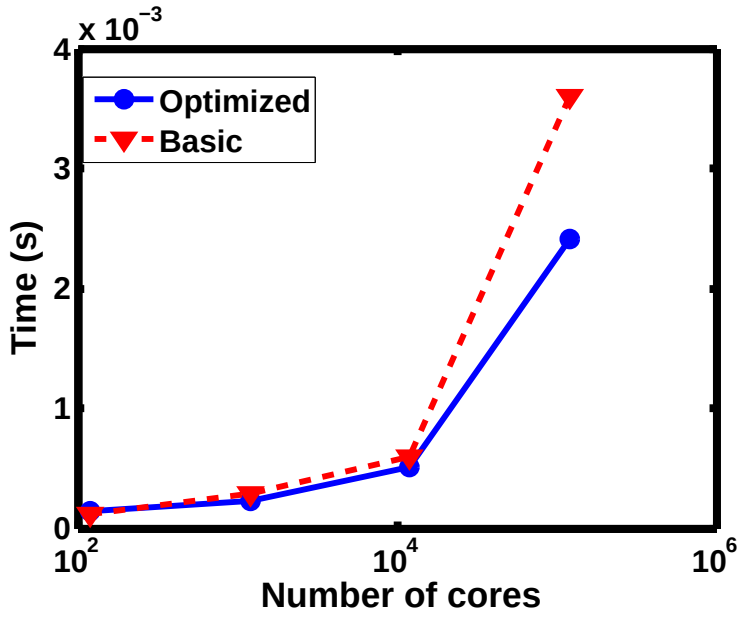

Figure 5.3: Comparison of optimized Alias method against the basic method with task size 8KB.

slower than the basic algorithm. Figure 5.3 shows that the optimized algorithm has much better performance than the basic algorithm for large core counts. It is close to 30% better with 120,000 cores, and 15-20% better with 1,200 and 12,000 cores.

We next analyze the reason for the improved performance. Figure 5.4 considers the average over all runs for the maximum time taken by different components of the algorithm. As with the analysis of the basic algorithm, the total maximum time is smaller than the sum of the maximum times of each component. We can see that the wait time is smaller than that of the basic algorithm shown in figure 5.1, which was the purpose of this optimization. The improvement is around 60% with 120,000 cores and 20% on 12,000 cores. Surprisingly, the MPI_Allgather time also reduces by around 30% on 120,000 cores and 20% on 12,000 cores. It appears that the MPI implementation does not optimize for the

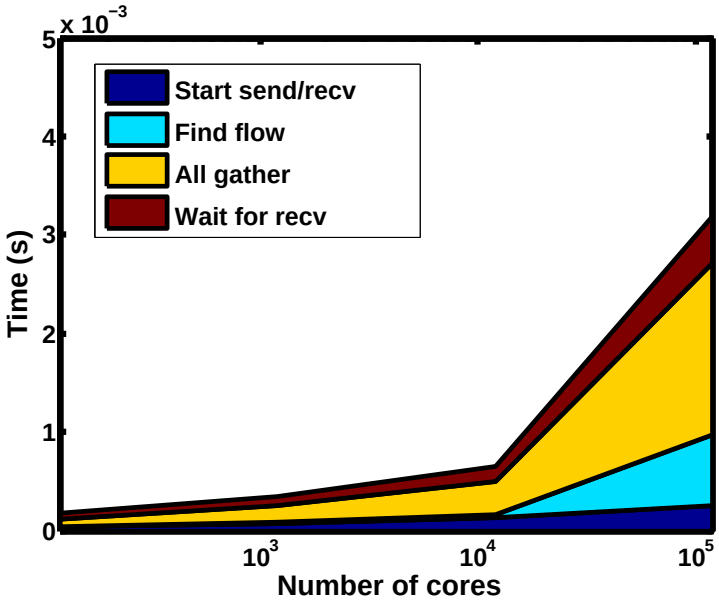

Figure 5.4: Maximum time taken for different components of the optimized Alias method with task size 8KB.

topology of the nodes that are actually allocated for a run, and instead uses process ranks. The ranks specified by this algorithm happens to be good for the MPI_Allgather algorithm. This improvement depends on the nodes allocated. In the above experiment, with 120,000 cores, the set of allocated nodes consisted of six connected components. In a different run, we obtained one single connected component. The use of MPI_Allgather with the optimized algorithm did not provide any benefit in that case. It is possible that the MPI implementation optimized its communication routines under the assumption of a single large piece of the torus. When this assumption is not satisfied, perhaps its performance is not that good.

The performance gains are smaller with smaller core counts, which can be explained by the following observations. Figure 1.1 shows the node allocation for the 12,000 core run. We can see that we get a large

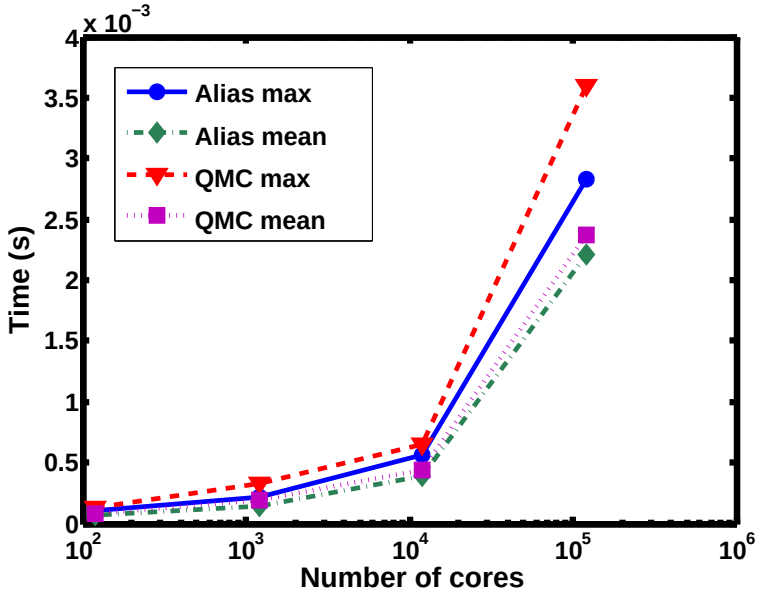

Figure 5.5: Comparison of the optimized Alias algorithm against the existing QWalk implementation with task size 672B.

number of connected components. Thus, inter-job contention can play an important role. Each component is also not shaped close to a cube. Instead, we have several lines and 2-D planes, long in the z direction. This makes it hard to avoid intra-job contention, because each node is effectively using fewer links, making contention for links more likely. It is perhaps worthwhile to consider improvements to the node allocation policy. For 120 and 1,200 cores, typically each connected component is a line (or a ring, due to the wrap-around connections), which would lead to contention if there were several messages sent. However, the number of nodes with imbalance is very small, and contention does not appear to affect performance in the load migration phase. Consequently, improvement in performance is limited to that obtained from the all-gather operation.

We next compare the optimized alias implementation against the

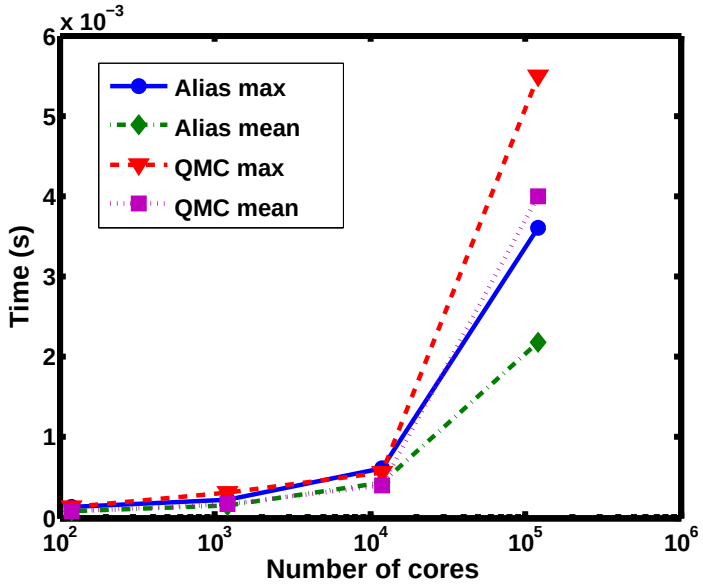

Figure 5.6: Comparison of the optimized Alias algorithm against the existing QWalk implementation with task size 2KB.

QWalk implementation in figures 5.5, 5.6, and 5.7. The new algorithm improves the performance by up to 30-35% in some cases, and is typically much better for large numbers of cores. The improved performance is often due to improvement in different components of the algorithm and its implementation: all-gather, task migration communication cost, and to a smaller extent, time for the flow computation. We can see from these figures that the time for 2KB tasks is higher on 120,000 cores than that for larger messages, especially with the QWalk algorithm. This was a consistent trend across the runs with QWalk. The higher time with the Alias method is primarily the result of a couple of runs taking much larger time than the others. These could, perhaps, be due to inter-job contention. We did not ignore this data as an outlier, because if such a phenomenon occurs 20% of the time, then we believe that we need to consider it as a reality of the computations in realistic conditions.

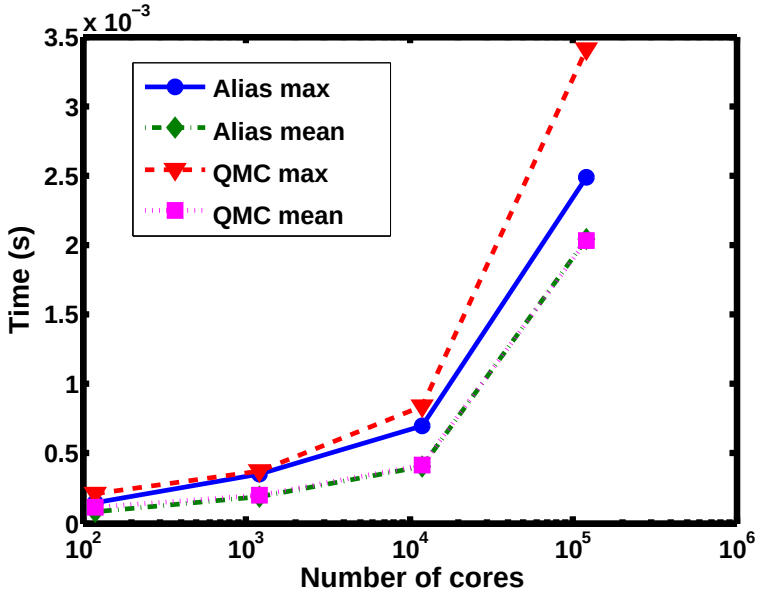

Figure 5.7: Comparison of the Alias algorithm with the existing QWalk implementation with task size 32KB.

We know that the alias method is optimal in the maximum number of messages received by any node, and found (not shown) that QWalk requires an increasing maximum number of receives with increasing core count. However, when we measure the mean number of tasks sent per core, figure 5.8, we find that QWalk is optimal. The alias method is approximately a factor of two worse in terms of the number of messages sent. Although we do not see this in tests with realistic message sizes, for sufficiently large messages it is clear that there must be a cross-over in the preferred load balancing algorithm. At some point the existing QWalk algorithm will be preferred since the communications will be bandwidth bound.

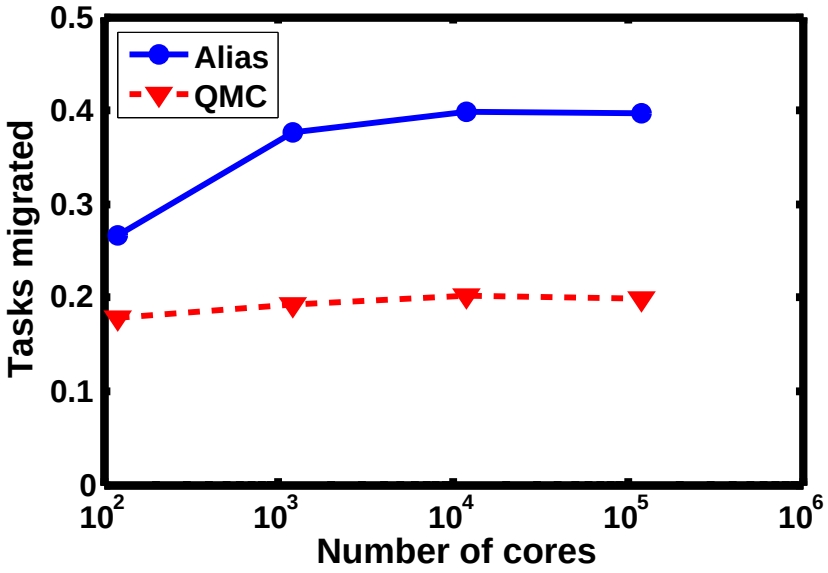

Figure 5.8: Mean number of tasks sent per core.

5.5 Summary

We showed that suitable assignment of tasks to cores on large machines such as Jaguar reduced the communication overhead significantly. We demonstrated this using a new load balancing algorithm that we have developed. We optimized the implementation with a selective mapping determined based on the allocation topology and communication pattern, and demonstrated that it has better performance due to reduced network contention. The relative performance of the algorithm to existing methods is expected to increase with the increased compute element count of upcoming machines.

6. Optimization of the Hop-Byte Metric

6.1 Problem Formulation

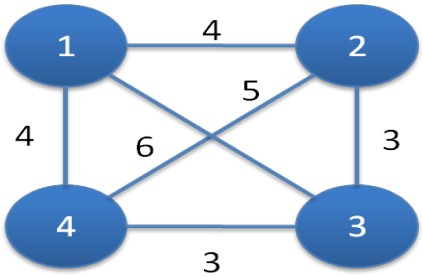

Figure 6.1: Node graph.

We model the problem using two graphs. The node graph G, such as in figure 6.1 is an undirected graph with vertices representing the nodes of the machine that have been allocated to the job submitted, and edge weights e_{ij} representing the number of hops between the vertices i and j linked by that edge. The hops can be determined from the machine topology information and knowledge of the nodes actually allocated. These can usually be obtained on most supercomputing systems. We assume that static routing is used, which is fairly common.

The task graph $G' = (V', E')$, such as in figure 6.2, represents the

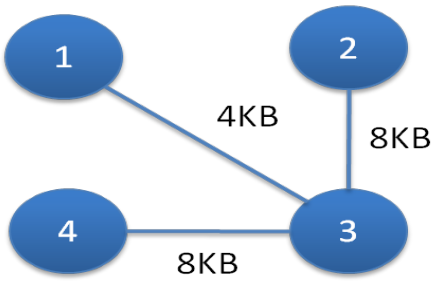

Figure 6.2: Task graph.

submitted job. Each vertex represents a process running on a node and edge weight e'_{ij} represents the total sizes of messages, in both directions, between vertices i and j linked by that edge. The number of vertices in this graph must equal the number in the node graph. If there are more tasks than nodes, then a graph partitioning algorithm can be applied to aggregate tasks so that this condition is satisfied.

Minimizing the hop-bytes then is the following quadratic assignment problem, where x_{ij} = 1 implies that task j is assigned to node i.

$$min \sum_{ij} \sum_{kl} e_{ik} e'_{jl} x_{ij} x_{kl}, \quad (6.1)$$

subject to:

$$\sum_{i} x_{ij} = 1, for\ all\ j$$

$$\sum_{j} x_{ij} = 1, for\ all\ i$$

x_{ij} in {0,1}

This is a well known NP hard problem and considered hard to approximate, though there are reasonable approximation algorithms [78] for dense instances. The exact solution can be found for small instances using branch and bound (bounds are used to reduce the search space). A couple of popular lower bounds are the elimination bound and the Gilmore-Lawler bound. We use the exact solution for small instances, and also the bounds for medium size instances, in order to evaluate the effectiveness of our heuristics.

6.2 Mapping Heuristics

6.2.1 GRASP Heuristic

Several heuristics have been proposed for QAP, based on meta-heuristics such as simulated annealing, tabu search, ant colony optimization, and greedy randomized adaptive search procedures (GRASP). GRASP is based on generating several random initial solutions, finding local optima close to each one, and choosing the best one. We use a GRASP heuristic for QAP from [77].

6.2.2 MAHD and Exhaustive MAHD Heuristics

We first describe our faster heuristic, Minimum Average Hop Distance (MAHD), in Algorithm 1 below. It improves on the following limitation of the GGE [13] heuristic. GGE replaces step 7 of algorithm 1 with a strategy that places the task on the node closest to its most recently mapped neighbor. We would ideally like this task to be close to all its neighbors. MHT [12] addresses this by placing the node closest to the centroid of all previously mapped neighbors. On the other hand MHT works only on

meshes. Our algorithm works on a general graph, and places the task, in this step, on the node that has the minimum average hop distance to nodes on which all previously mapped neighbors of the task have been mapped. MHT also selects a random node on which to place the initial task. We intuitively expect a task with the maximum number of neighbors to be a "central" vertex in a graph, and so try to map it to a node which is "central" in its graph. We do this by placing it on the node with the minimum average hop distance to any other vertex. The first task selected may not actually be "central" (for instance, in the sense centrality measures such as betweenness centrality). We introduce an Exhaustive MAHD (EMAHD) heuristic to see if a better choice of initial vertex is likely to lead to significant improvement in mapping quality. In this heuristic, we try all possible nodes as starting vertices, and then choose the one that yields the best mapping quality.

Algorithm 1 MAHD(G, G')

1. s = vertex in G with maximum number of neighbors
2. p = vertex in G' with minimum average hop distance to all other vertices
3. Assign task s to node p
4. Insert all neighbors of s into max-heap H, where the heap is organized by the number of neighbors
5. **while** H in not empty **do**
6. s = H.pop();
7. s is mapped to a node with minimum average hop distance to processes hosting mapped neighbors of s;
8. Insert neighbors of s into H if they are not in H and have not been mapped;
9. **end while**

6.2.3 Hybrid Heuristic with Graph Partitioning

If $|V'|$ is too large for GRASP to be feasible, then we use the following hybrid heuristic. We partition graphs G and G' into partitions of size p each. Any graph partitioning algorithm can be used. We use a multilevel heuristic available in parMetis. We create graphs H and H' corresponding to the partitions of G and G' respectively. In H, each vertex corresponds to a partition in G and in H', each vertex corresponds to a partition in G'. Each edge in H has weight corresponding to the average hops from nodes between the two partitions linked by that edge. Each edge in H' has weight corresponding to the total message sizes between the two partitions linked by that edge. A mapping of partitions in H' to partitions in H is performed using GRASP and mapping of tasks to nodes within each partitions is again performed using GRASP with corresponding subgraphs of G and G'.

6.3 Evaluation of Heuristics

6.3.1 Experimental Platform

The experimental platform is the Cray XT5 Kraken supercomputer at NICS. We used the native Cray compiler with optimization flag "$-O3$". The QAP codes were obtained from QAPlib (http://www.opt.math.tu-graz.ac.at/qaplib/codes.html), which includes codes from a variety of sources. A branch and bound algorithm was used for the exact solution, the Gilmore-Lawler bound for a lower bound[1] and a dense GRASP heuristic for larger problem sizes.

[1]The Gilmore-Lawler bound performed better than the elimination bound for large problem sizes.

Three standard collective communication patterns used in MPI implementations – Recursive Doubling, Bruck, and Binomial Tree – were studied. We also used the following three irregular communication patterns. A 3D Spectral element elastic wave modeling problem (3DSpec) and a 2D PDE (Aug2dc) from the University of Florida sparse matrix collection, and a 2D unstructured mesh pattern (Mesh) from the ParFUM framework available in CHARM++ library.

OSU MPI micro benchmark suite was used for the empirical tests on the Kraken machine to observe the impact of the heuristic based mapping on MPI collective calls.

6.3.2 Experimental Results

Figures 6.3, 6.4 compare the default mapping, GRASP result, and the Gilmore-Lawler bound for small problem sizes. The quality (hop-byte metric value) is divided by that for the exact solution to yield a normalized quality.

ProblemSize	GRASP	EMAHD	Default	GGE	MAHD
8	1.43	1.48	2.22	2.18	1.83
16	2.98	3.76	5.36	3.88	4.02
32	2.94	4.05	5.88	5.19	4.30
64	4.34	4.72	8.19	6.04	5.43
100	2.56	2.74	3.84	3.86	3.16
128	2.92	2.91	3.49	4.21	3.51
144	2.73	2.61	4.25	4.16	3.22
192	3.73	3.56	5.12	5.37	4.06
200	3.14	2.77	4.30	4.05	2.83
216	1.84	1.73	2.99	2.83	1.90
250	3.66	3.39	5.50	5.37	3.45
300	3.42	3.29	5.66	5.16	3.73

Table 6.1: Hops per byte.

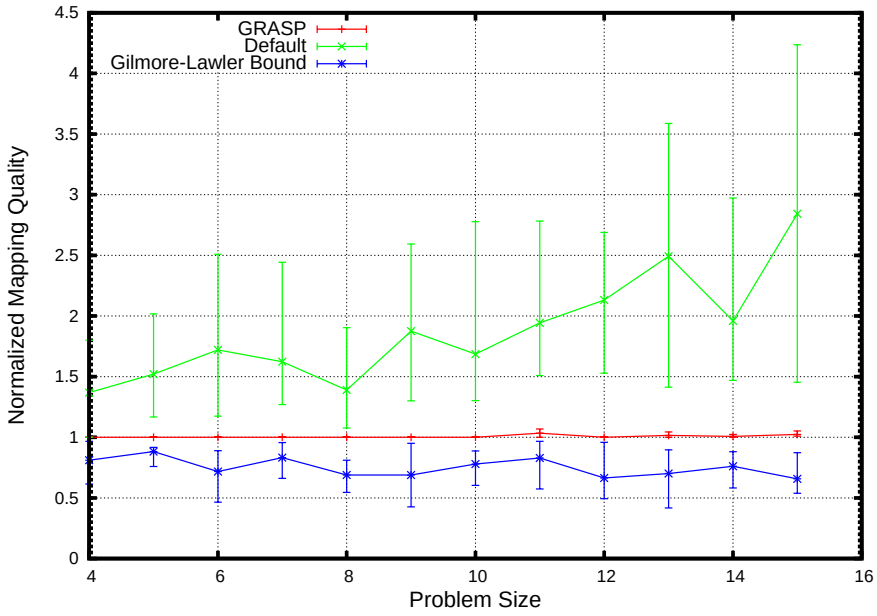

Figure 6.3: Quality of solutions on the Recursive Doubling pattern for small problem sizes.

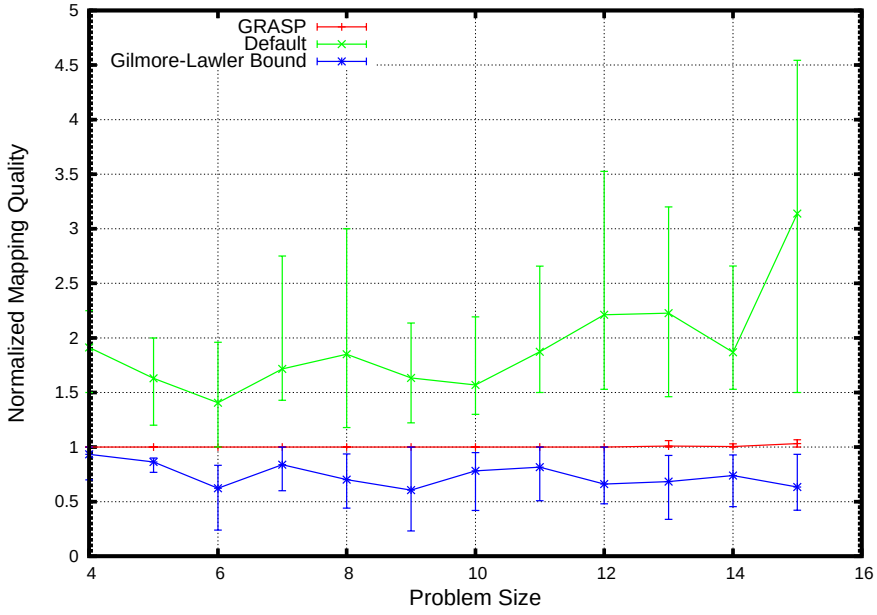

Figure 6.4: Quality of solutions on the Binomial Tree pattern for small problem sizes.

We can see that the GRASP is close to the exact solution for these problem sizes. We also note that the lower bound is a little over the half of the exact solutions toward the higher end of this size range. Similar trend were observed for the other patterns, which are not shown here.

The table 6.1 compares the hops per byte metric value for different problem sizes while mapping the Mesh pattern. Figures 6.5-6.10 compare the heuristics for medium problem sizes. The quality is normalized against the default solution, because computing with the exact solution is not feasible. These figures, therefore, show improvement over the default mapping.

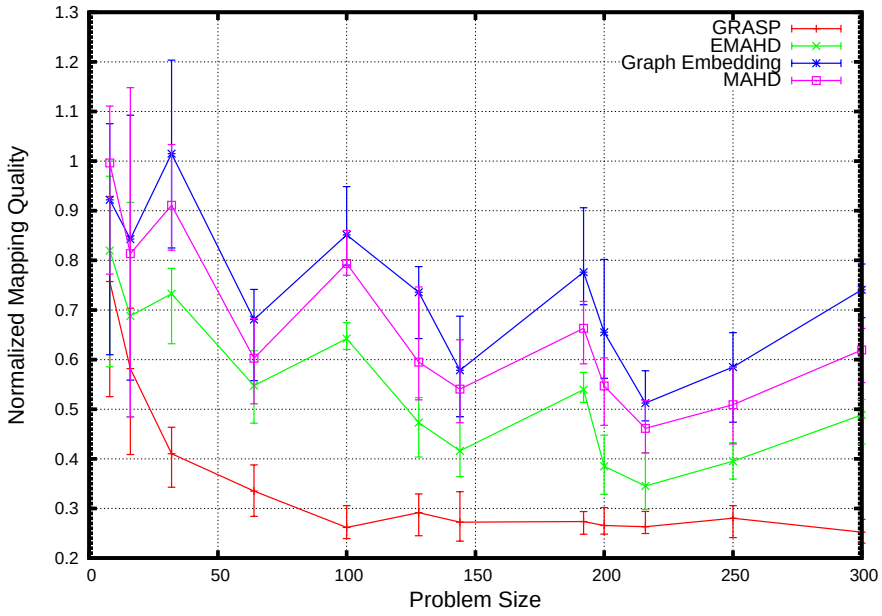

Figure 6.5: Quality of solution on the Recursive Doubling pattern for medium problem sizes.

We can see that the GRASP heuristic is consistently better than the default and the GGE heuristic, while MAHD and EMAHD are sometimes comparable to GRASP. EMAHD is often much better than MAHD, suggesting that a better choice of the initial vertex has potential to make

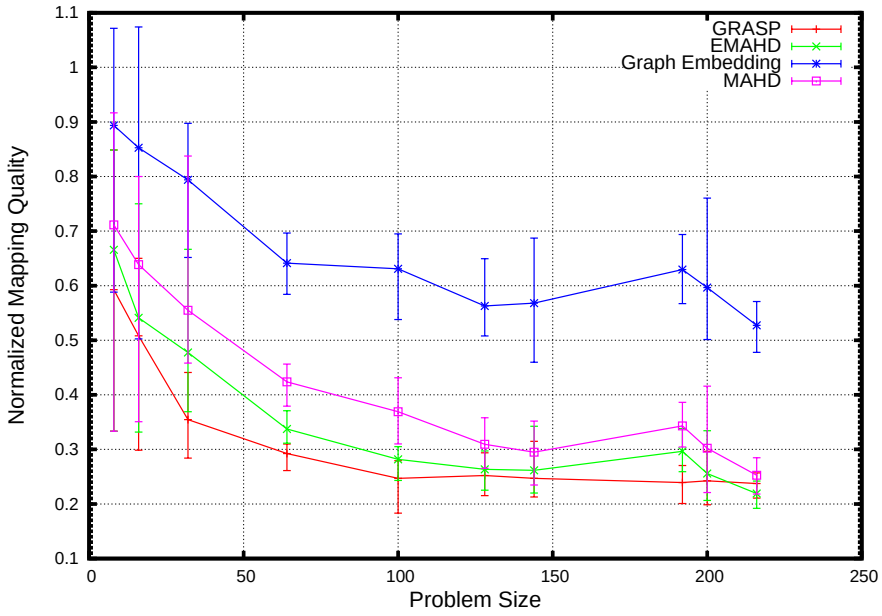

Figure 6.6: Quality of solution on the Binomial Tree pattern for medium problem sizes.

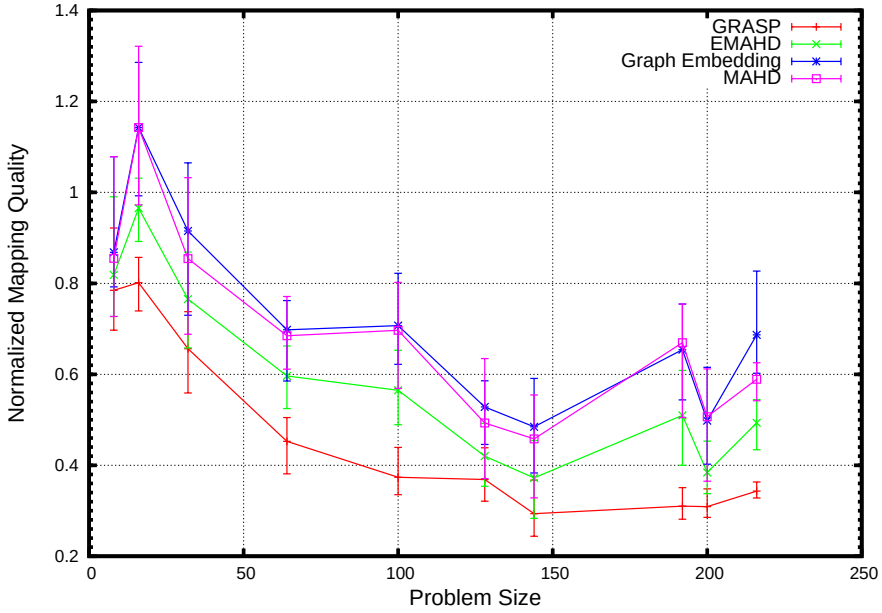

Figure 6.7: Quality of solution on the Bruck pattern for medium problem sizes.

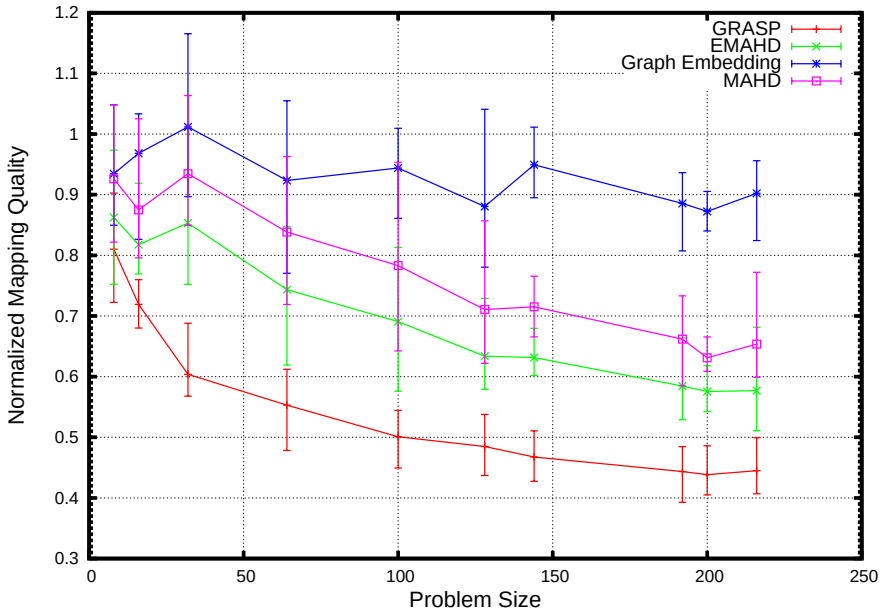

Figure 6.8: Quality of solution on the 3D Spectral pattern for medium problem sizes.

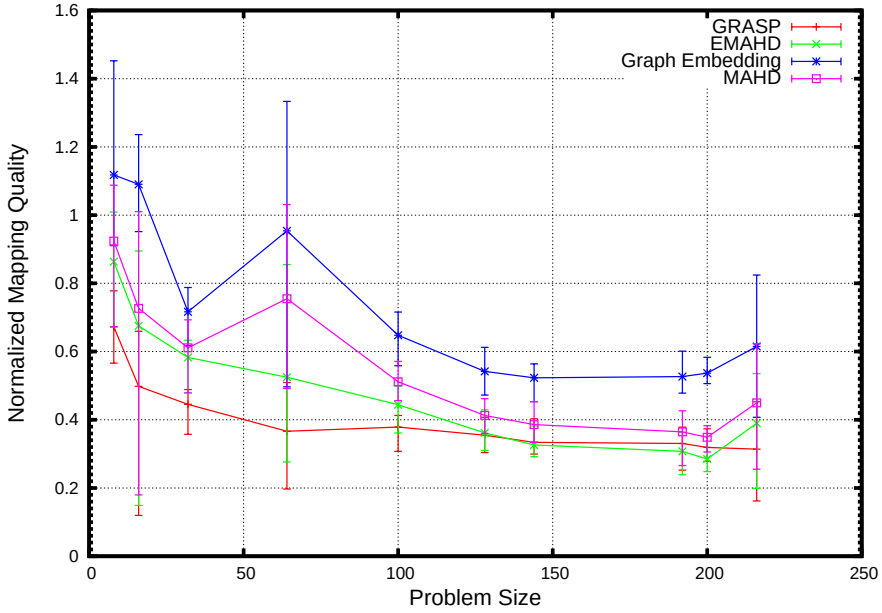

Figure 6.9: Quality of solution on the Aug2D pattern for medium problem sizes.

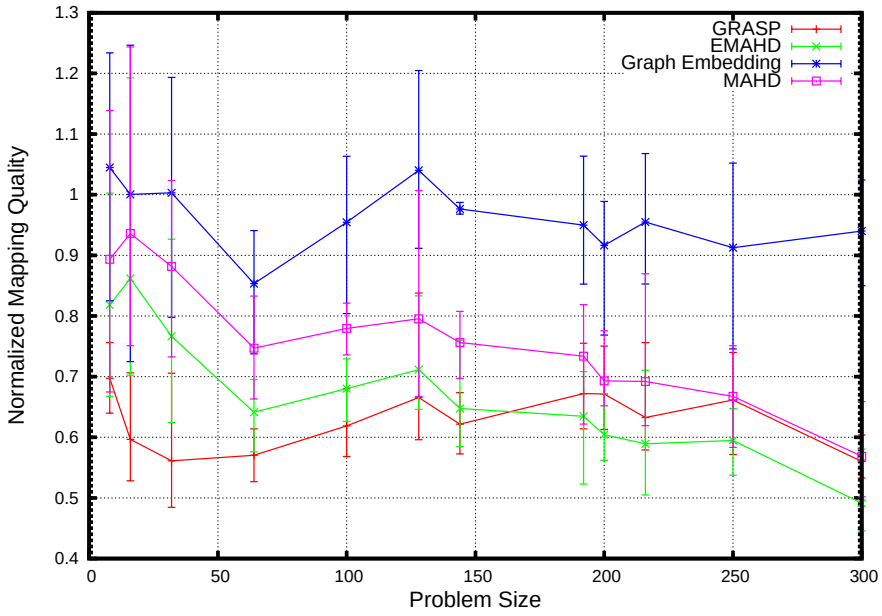

Figure 6.10: Quality of solution on the Mesh pattern for medium problem sizes.

significant improvement to MAHD.

However, the time taken by GRASP and EMAHD are significantly larger than that for GGE or MAHD, as shown in figure 6.11. Consequently, they are more suited to static communication patterns. Since EMAHD typically does not produce better quality than GRASP either, it does not appear very useful. On the other hand, its quality suggests that if a good starting vertex can be found for MAHD without much overhead, then MAHD's quality can be improved without increasing its run time. When the communication pattern changes dynamically, then MAHD is a better alternative to the above two schemes and also to GGE. It is as fast as GGE, while producing mappings of better quality. Its speed also makes it feasible to use it dynamically, while GRASP is too slow.

An alternative to MAHD for large problem sizes with dynamic communication patterns is the hybrid algorithm. Preliminary results

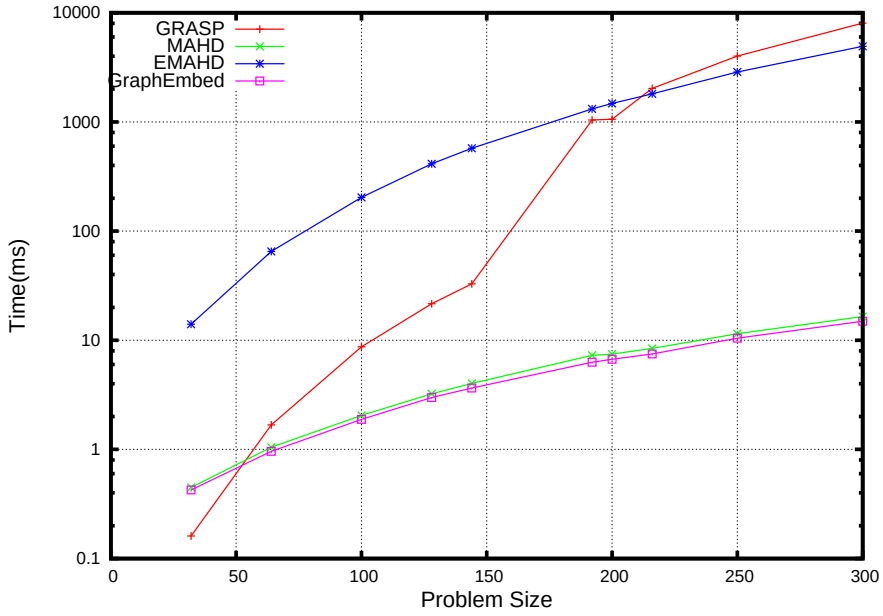

Figure 6.11: Comparison of time taken by the heuristics.

on 1000 nodes with partitions of size 125 are shown in figure 6.12. The hybrid algorithm performs better than the default and GGE with both communication patterns. It is better than MAHD and EMAHD for Recursive Doubling, but is worse with the Binomial Tree. The Binomial Tree has less communication volume than Recursive Doubling, and further experiments are necessary to check if the hybrid algorithm tends to perform better when the communication volume is larger. We note that even for medium sized problems, GRASP (which is the underlying heuristic behind the hybrid algorithm), was comparable with EMAHD and MAHD for the Binomial Tree, but much better for Recursive Doubling. The hybrid scheme produces a further reduction in quality, which makes it worse than MAHD and EMAHD for Binomial Tree, but since GRASP is much better for Recursive Doubling, the hybrid algorithm is better than MAHD and EMAHD for it, though by a smaller margin.

As the number of partitions increases, the hybrid algorithms relative advantage decreases, as can be seen in figure 6.13[2], which is based on 16 partitions of size 125 each.

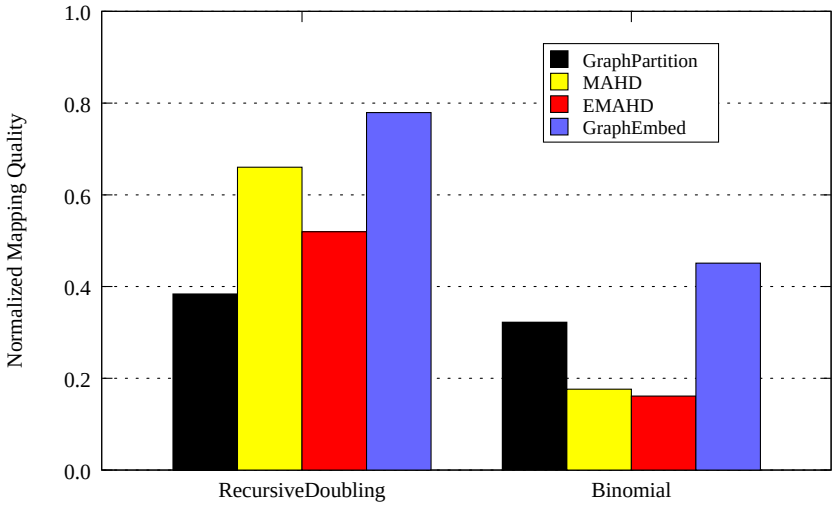

Figure 6.12: Comparison of heuristics on 1000 nodes (12,000 cores).

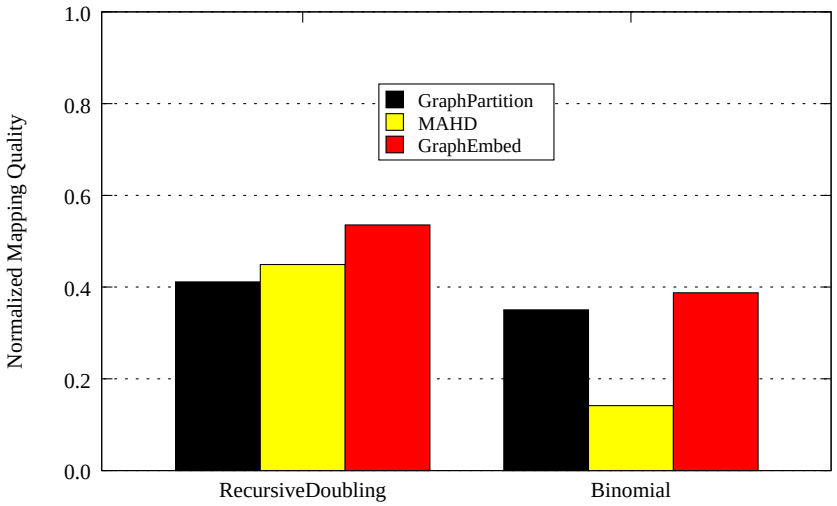

Figure 6.13: Comparison of heuristics on 2000 nodes (24,000 cores).

We next evaluate how well GRASP compares with the lower bound

[2]It was not feasible to use EMAHD for 2000 nodes due to the time required by it.

for medium problem sizes. Figures 6.14-6.19 show that GRASP is around a factor of two from the Gilmore-Lawler bound. As shown earlier, the above bound was usually a little higher than half the exact solution for small problem sizes, and did not get tighter with increased sizes. These results, therefore, suggest that GRASP is close to the optimal solution.

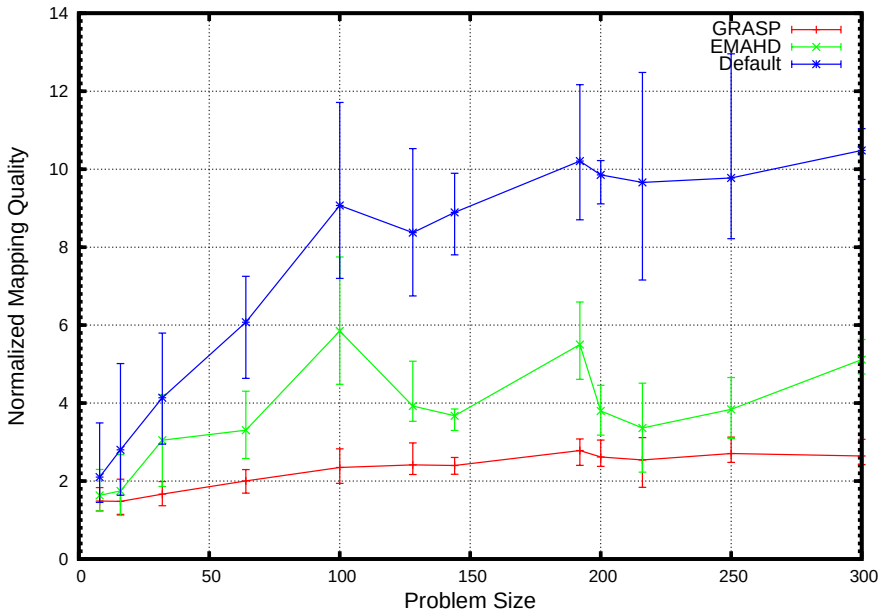

Figure 6.14: Quality of solution on the Recursive Doubling pattern compared with a lower bound.

Finally, we wish to verify if improving the hop-byte metric actually improves the communication performance. Preliminary studies with Recursive Doubling on the MPI_Allgather implementation with 1KB messages on problems with 128 nodes showed that GRASP and EMAHD are about 25% faster than the default and 20% faster than GGE. The improvement over the default is significant, though not as large as that indicated by the hop-byte metric, because that metric is only an indirect indication of the quality of the mapping. However, it does suggest that optimizing the hop-byte metric leads to improved performance.

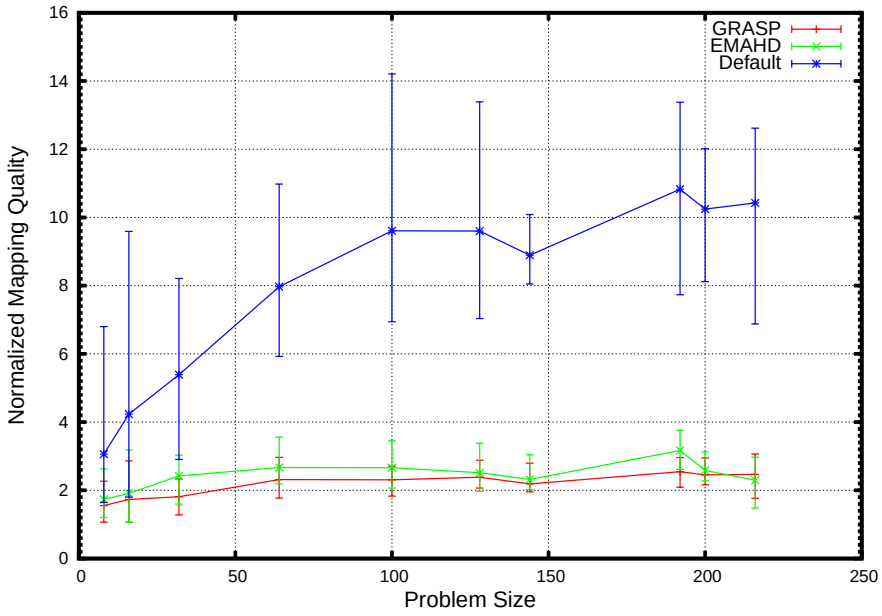

Figure 6.15: Quality of solution on the Binomial Tree pattern compared with a lower bound.

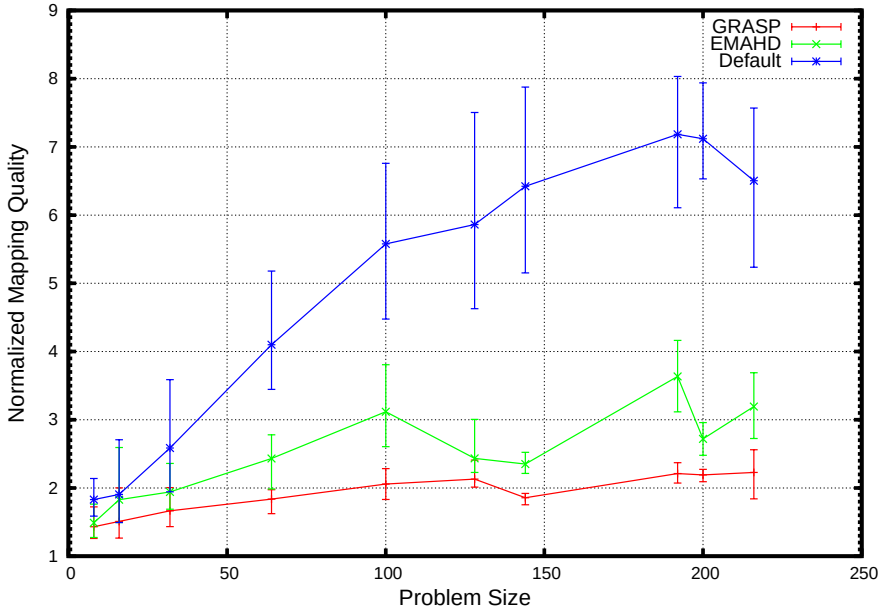

Figure 6.16: Quality of solution on the Bruck pattern compared with a lower bound.

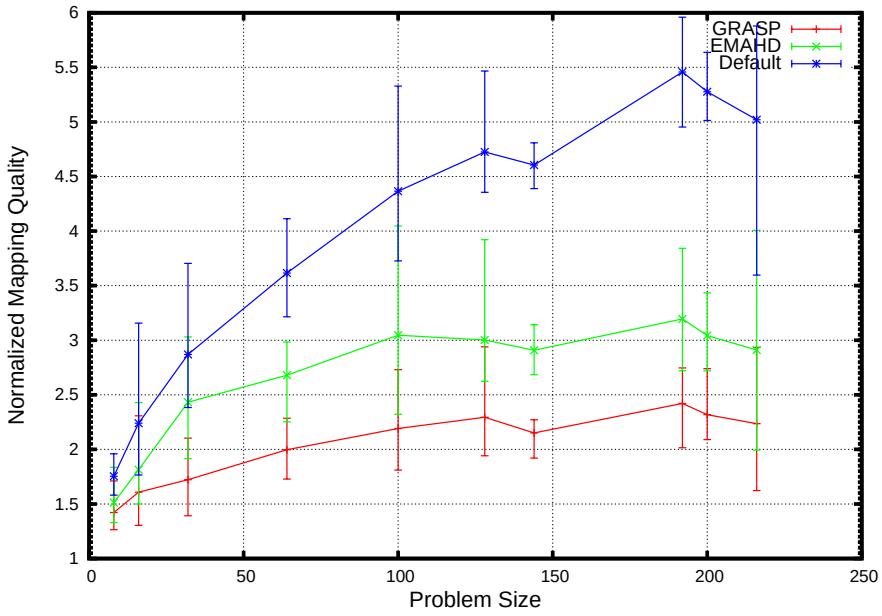

Figure 6.17: Quality of solution on the 3D Spectral pattern compared with a lower bound.

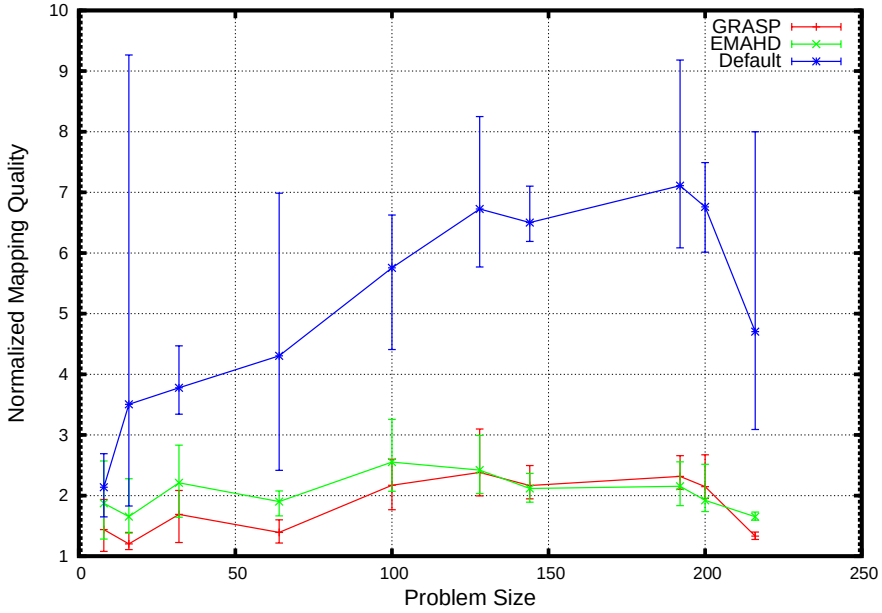

Figure 6.18: Quality of solution on the Aug2D pattern compared with a lower bound.

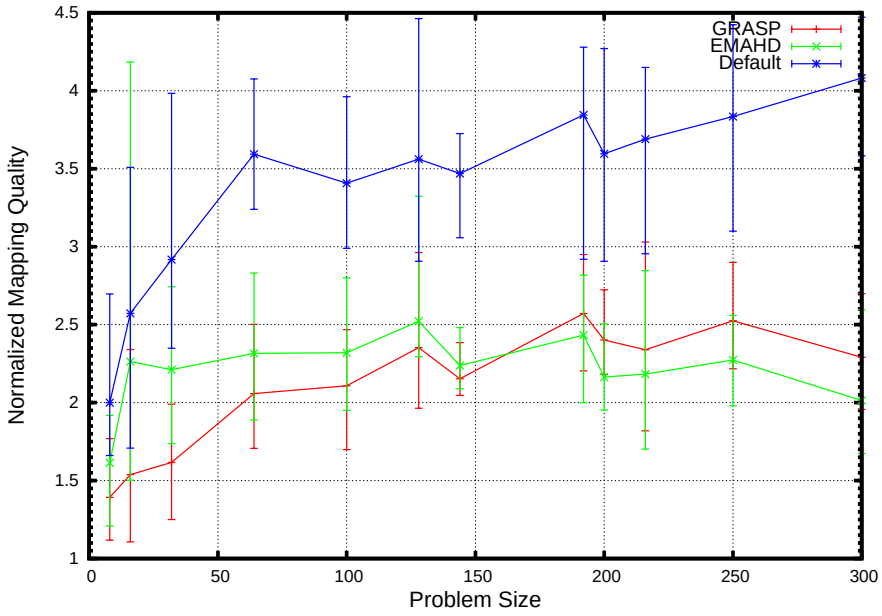

Figure 6.19: Quality of solution on the Mesh pattern compared with a lower bound.

6.4 Summary

We have shown that optimizing for the hop-bytes metric using the GRASP heuristic leads to a better mapping than existing methods, which typically use some metric just to evaluate the heuristic, rather than to guide the optimization. We have evaluated the heuristic on realistic node allocations, which typically consist of many disjoint connected components. GRASP performs better than GGE, which is among the best prior heuristics that can be applied to arbitrary graphs with arbitrary communication patterns, and performs much better than the default mapping. In fact, GRASP is optimal for small problem sizes. Comparison with the lower bound suggests that GRASP may be close to optimal for medium problem sizes too. However, it does not scale well with problem size and is infeasible for large graphs. We proposed two

solutions for this. One is the MAHD algorithm and the other is a hybrid algorithm. The former is fast and reasonably good, while the latter is sometimes better, but much slower than MAHD. For static communication patterns on medium sized graphs, GRASP would be the best option. For large problems with static communication patterns, the hybrid approach would be a good alternative, especially when the communication volume is large. However, MAHD too can be effective. For dynamic communication patterns, MAHD is the best alternative. In fact, results with EMAHD suggest that if a good starting vertex can be found, then MAHD may be competitive even for many static patterns. MAHD takes roughly the same time as GGE, but consistently outperforms it. Preliminary experiments on MPI also suggest that optimizing for the hop-byte metric improves the actual MPI collective communication latency, though not to the extent predicted by this metric. This is reasonable, because the metric does not directly account for the congestion bottleneck.

One direction for future work is in reducing the time taken by the GRASP heuristic. GRASP is a general solution strategy, rather than a specific implementation. The particular implementation that we used is for a general QAP problem. We plan to develop an implementation specific to our mapping problem. For instance, solutions generated by the fast heuristics can be used as starting points in GRASP, thereby reducing the search space. Furthermore, we used a dense GRASP implementation because the node graph is complete. We can remove edges with heavy weights (corresponding to nodes that are far away) so that a sparse algorithm can be used. A different direction lies in optimizing for a different metric. The actual bottleneck is contention on

specific links. We have posed the problem of minimizing the maximum contention as an integer programming problem, and are developing heuristics to solve it. Some of the preliminary results are presented in the next chapter.

7. Maximum Contention Metric

7.1 Problem Formulation

Another practical metric that can be used to evaluate the quality of mapping is the maximum contention metric. The hop-bytes metric gives an indication of the average communication load on each link in the network. However, in reality, the communication bottleneck is normally the bottleneck link, having the largest traffic on it. We will ignore inter-job contention for the time being, and consider only intra job contention. Our goal is to assign tasks to nodes in order to minimize the maximum traffic on any given link. Routing plays an important role in the contention experienced in the network; topology information is not sufficient. We will assume static routing, and that we have information on the links traversed by messages from any pair of nodes.

We can pose this problem with a linear objective function subject to quadratic constraints, as shown below. Here, d_{mijkl} can be computed as the size of the message from task i to task k, if the route from node j to node l uses link m, and zero if the route does not use that link. And t is the load on the link with maximum load. The constraint ($t \geq y_{\mathrm{m}}$) ensures that t is as high as the load on the link with maximum load. The other constraints ensure the assignment of each task to a distinct node. This

problem too is NP hard [79]. The solution approaches to arrive at an approximate solution and their evaluation are discussed next.

$$min\ t, \quad (7.1)$$

subject to:

$$t >= y_m,\ for\, all\, m$$

$$y_m = \sum_{ijkl} d_{mijkl}\, x_{ij}\, x_{kl},\ for\, all\, m$$

$$\sum_{i} x_{ij} = 1, for\, all\, j$$

$$\sum_{j} x_{ij} = 1, for\, all\, i$$

x_{ij} in {0,1}

7.2 Heuristics and Their Evaluation

The heuristic solutions used for equation 6.1 can as well be used for this problem. Evaluate each of the heuristic with the maximum contention metric, and choose the best one. Our experimental results showed that EMAHD outperforms all the other heuristics for this metric.

The next step in improving the solution is to perform a local optimization on the mapping resulted with the EMAHD heuristic. The local optimization involves choosing pairs of tasks that can be swapped such that metric value improves. The results for this are referred in the figures and table as 'LocalOpt'. The table 7.2 compares the maximum contention metric value for different node allocation sizes while mapping the Mesh pattern.

ProblemSize	Default	GGE	MAHD	GRASP	EMAHD	LocalOpt
4	97	73	62	73	62	62
8	144	104	162	151	113	87
16	168	233	144	170	128	115
32	307	267	285	338	228	149
64	576	413	634	484	396	193
100	473	679	501	514	376	181
128	481	457	455	522	406	227
144	481	449	510	481	377	221
192	567	565	538	560	449	277
200	622	576	457	416	300	201
216	567	575	632	632	464	256

Table 7.1: Load on the maximum congested link.

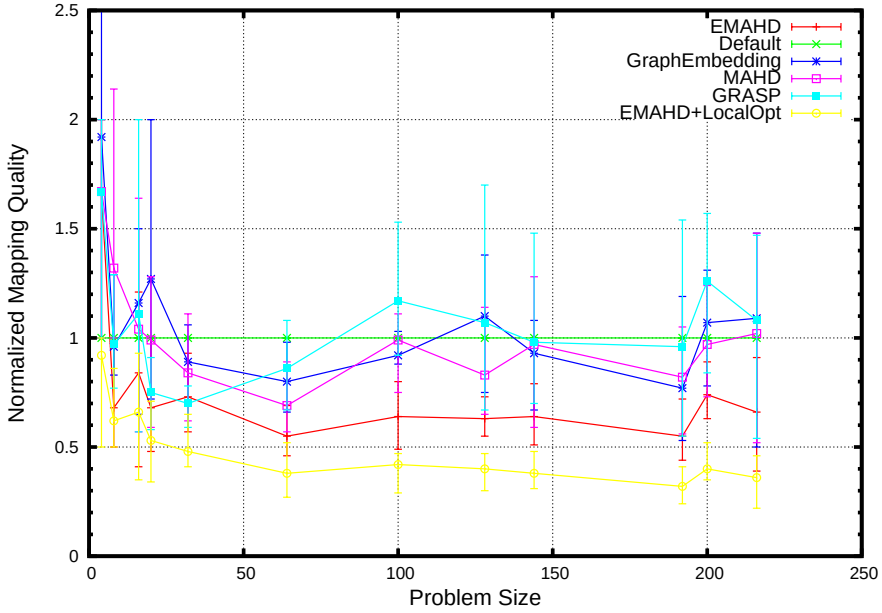

Figure 7.1: Quality of solution on the Recursive Doubling pattern for medium problem sizes.

Figures 7.1 to 7.6 compare heuristics for this metric on medium problem sizes. The quality is normalized against the default mapping solution. All the figures show that EMAHD consistently outperforms all the other heuristics. And idea of using the local optimization to improve

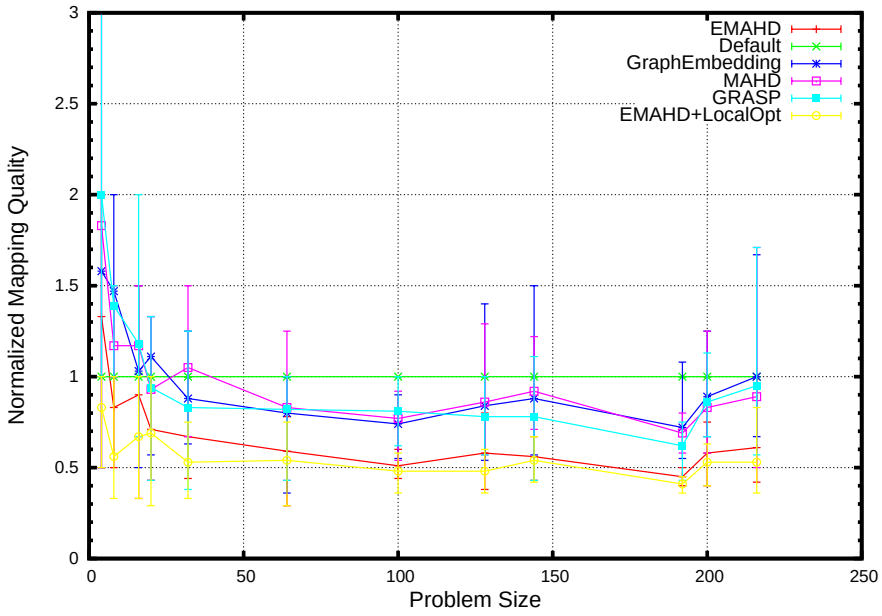

Figure 7.2: Quality of solution on the Binomial Tree pattern for medium problem sizes.

the EMHAD solution gives significantly better results. We can see that though GRASP outperformed other heuristics for the hop-byte metric, here EMAHD consistently gives better results over GRASP. The figure 7.3 shows that for the Bruck pattern, for most of the problem sizes, all the heuristics other than EMHAD provides mappings that worse than the default mapping.

The results show that EMAHD with local optimization gives a mapping quality that 100% to 120% better than the default mapping quality.

The above two approaches of solving equation 7.1 do not use the problem formulation. We will work on this by formulating this problem as a linear programming problem. This will produce a floating point solution, which we should be discretized in some reasonable manner to produce a final assignment. For example, we can use maximum

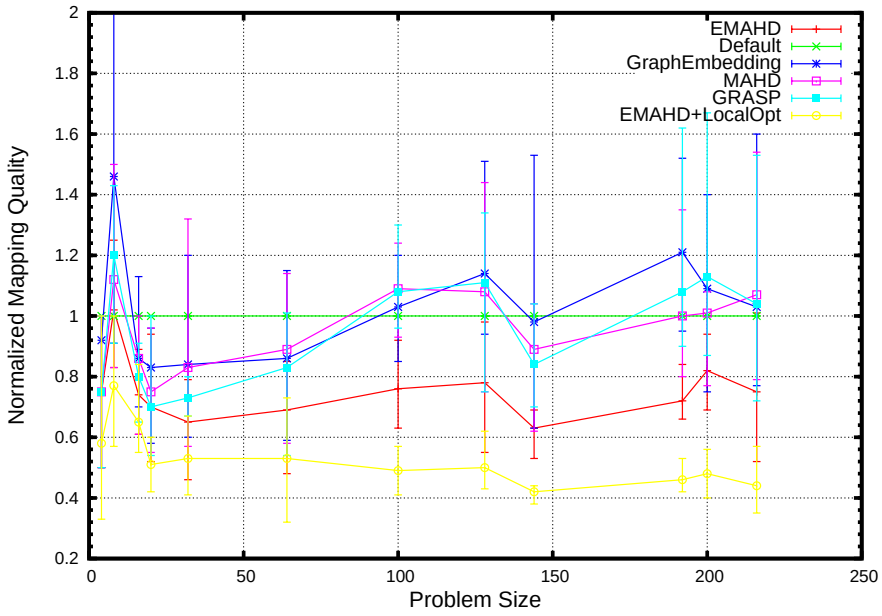

Figure 7.3: Quality of solution on the Bruck pattern for medium problem sizes.

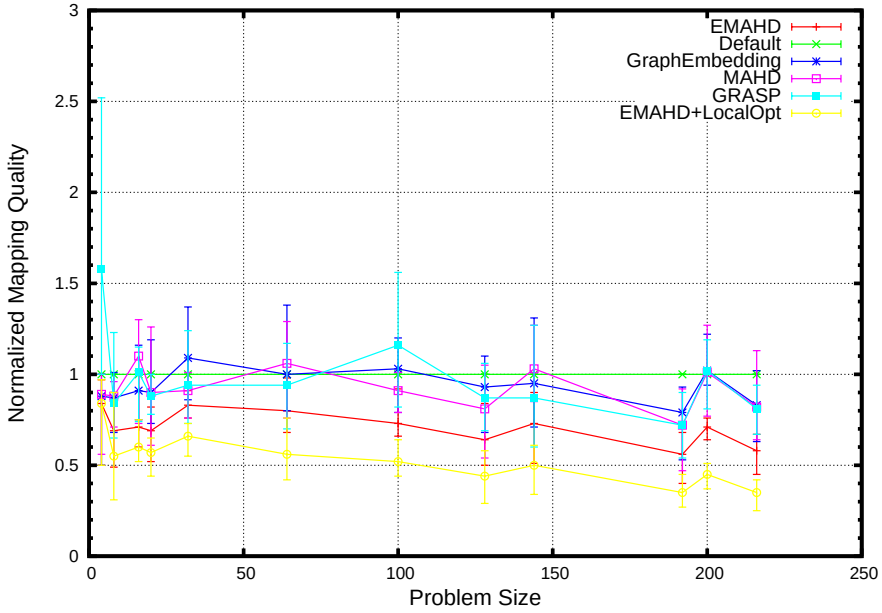

Figure 7.4: Quality of solution on the 3D Spectral pattern for medium problem sizes.

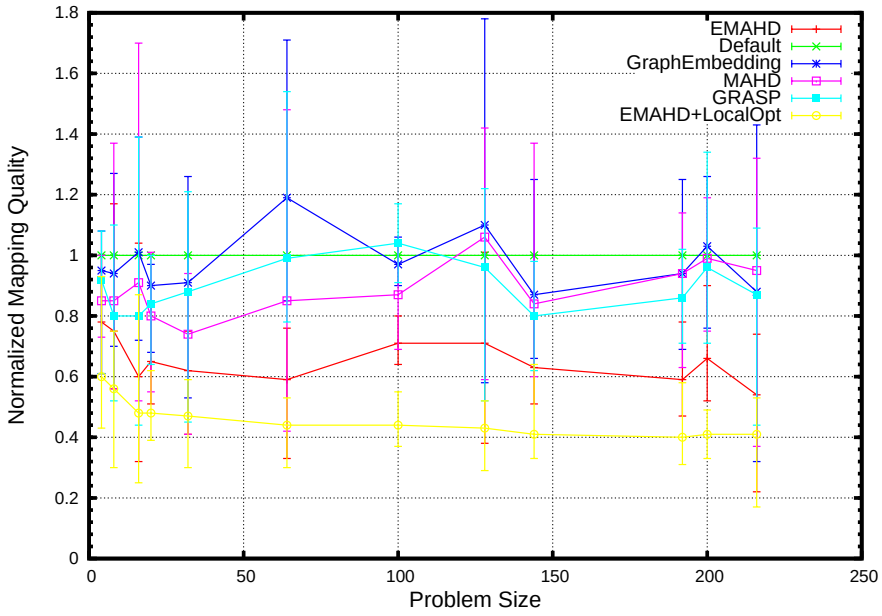

Figure 7.5: Quality of solution on the Aug2D pattern for medium problem sizes.

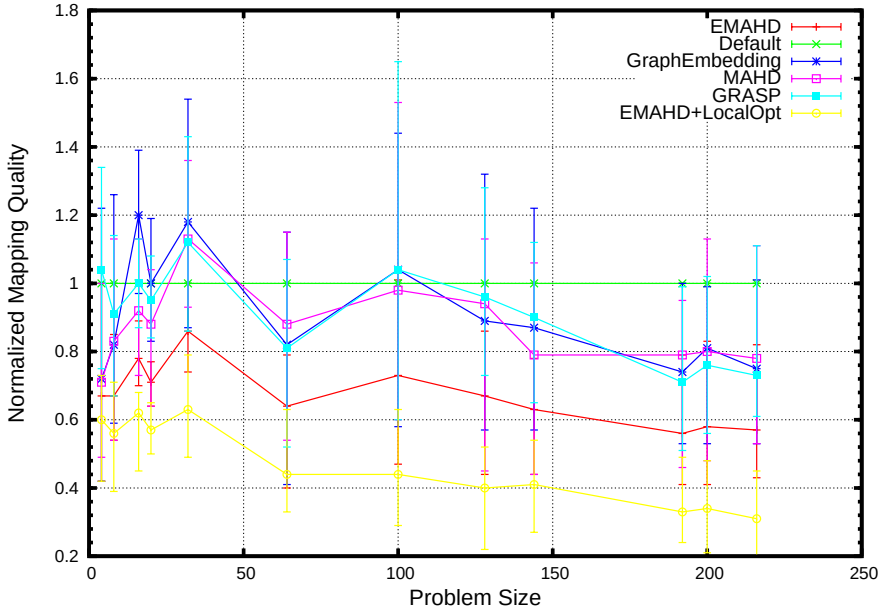

Figure 7.6: Quality of solution on the Mesh pattern for medium problem sizes.

Heuristic	Message Size	% of Improvement
EMAHD for Hop-bytes	128K	18.5
EMAHD for Hop-byte with local opt.	128K	24.1
EMAHD for Maximum contention	128K	23.7

Table 7.2: Percentage of improvement in bandwidth due to heuristic mappings over the bandwidth due to default mapping on 512 nodes.

matching on the obtained values. The solution to the linear programs will also give lower bounds on the optimal solution, which can be used to characterize the quality of the solution.

7.3 Empirical Evaluation

We wish to verify that if having an optimal maximum contention metric value actually improves the communication performance empirically. Preliminary experiments with the Recursive Doubling pattern for the MPI_Allgather with 1KB messages on problems with 128 nodes and 256 nodes showed that EMAHD for maximum contention metric with local optimization is about 15% to 25% faster than the default mapping and is consistently better than other heuristics though by a small margin. The EMAHD for maximum contention metric is better over the EMAHD for hop-bytes metric by 1.6% to 6% on 1024 nodes and by 1% to 13.8% on 512 nodes for the MPI_Allgather for less than 1K message sizes. For medium sized message sizes of size 128K, as shown in table 7.2, the EMAHD for maximum contention performs better than the default (identity) mapping by around 23%, and this improvement is better than improvement with the EMAHD for hop-bytes metric. The EMAHD for maximum contention metric with local optimization is not feasible to compute for larger problem sizes, hence results for that are

not presented here.

The improvement over the default is significant, though not as large as that indicated by the maximum contention metric. The next direction in this work is to develop heuristics to optimize this metric itself and to make the metric more complete by considering atleast approximately the inter job contention as well.

8. Conclusions and Future Work

The research conducted as part of this thesis reinstated the importance of topology and routing aware mapping in scaling the application performance on current supercomputers. Our results show that in spite of worm-hole routing, link contention can severely affect message latencies. Other important observation is that the nodes allocated by the job scheduler do not correspond to any standard topology, even when the machine does. The arbitrariness of the node allocations makes the mapping problem more interesting.

We have used topology and routing aware mapping to improve the performance of communication in the load balancing of QMC application on Jaguar. We were able to reduce the communication time in the load balancing phase by 60% with 120,000 cores and 20% on 12,000 cores, and this mapping also reduced MPI_allgather time by a similar amount. Our new dynamic load balancing algorithm developed can be used for computations with independent identical tasks, and the algorithm has some good theoretical properties. We have shown that it performs better than the current methods used in QMC codes in empirical tests.

In order to generalize the mapping techniques, we posed the mapping problem with the hop-byte metric as a quadratic assignment

problem and used a heuristic to directly optimize for this metric. We have shown that using the GRASP heuristic leads to a better mapping than existing methods, which typically use some metric just to evaluate the heuristic, rather than to guide the optimization. We evaluated our approach on realistic node allocations obtained on the Kraken system. Our approach yields values for the metric that are up to 75% lower than the default mapping and 66% lower than existing heuristics.

Preliminary experiments suggest that optimizing for the hop-byte metric improves the actual MPI collective communication latency, though not to the extent predicted by this metric. This is reasonable, because the metric does not directly account for the congestion bottleneck. One direction for future work is in reducing the time taken by the GRASP heuristic. GRASP is a general solution strategy, rather than a specific implementation. The particular implementation that we used is for a general QAP problem. We can develop an implementation specific to our mapping problem. For instance, solutions generated by the fast heuristics can be used as starting points in GRASP, thereby reducing the search space. Furthermore, we used a dense GRASP implementation because the node graph is complete. We can remove edges with heavy weights (corresponding to nodes that are far away) so that a sparse algorithm can be used.

A different direction lies in optimizing for a different metric. The actual bottleneck is contention on specific links. We have posed the problem of minimizing the maximum contention as an integer programming problem, and have developed heuristics to solve it. The preliminary results for this metric are encouraging.

The use of on-chip interconnect on some of the current and future

multi-core processors resulted in the importance of mapping even within a node. On a Cell processor, we observed that the SPE-thread affinity has a significant effect on inter-SPE communication throughput. The performance with an optimal affinity is a factor of two over the performance with default assignment. We developed a tool that automatically determines the ideal mapping when given a communication pattern. Using this tool on a particle transport application showed a difference in total application performance of over 10% between the best and worst mappings.

Bibliography

[1] Cray Inc. Cray XK Specifications. http://www.cray.com/Products/XE/Specifications/Specifications-XE6.aspx, 2012

[2] Cray Inc. Cray XK Specifications. http://www.cray.com/Products/XK6/Specifications.aspx, 2012

[3] IBM Blue Gene/Q Data Sheet. http://public.dhe.ibm.com/common/ssi/ecm/en/dcd12345usen/DCD12345USEN.PDF

[4] H. Yu, I. Chung, J. Moreira. Topology Mapping for Blue Gene/L Supercomputer. *ACM/IEEE conference on Supercomputing*, SC 06, 2006.

[5] T. Hoefler, T. Schneider. A. Lumsdain. Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks. *IEEE International Conference on Cluster Computing*, pages 116-125, 2008.

[6] F. Ercal, J. Ramanujam, P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. *Journal of Parallel and Distributed Computing*, 10, 1, 3544, 1990.

[7] S. Moh, C. Yu, D. Han, H. Y. Youn, B. Lee. Mapping strategies for switch-based cluster systems of irregular topology. In 8th *IEEE International Conference on Parallel and Distributed Systems*. 2001.

[8] E. Ma, L. Tao. Embeddings among meshes and tori. *Journal of Parallel and Distributed Computing*, 18, 4455, 1993.

[9] G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J.C. Sexton, R. Walkup. Opti-
mizing Task Layout on the Blue Gene/L Supercomputer. *IBM Journal of Research and Development*, Vol. 49 (2005) 489-500.

[10] P. Balaji, R. Gupta, A. Vishnu, P. Beckman. Mapping Communication Layouts to Network Hardware Characteristics on Massive-Scale Blue Gene Systems. *Comput. Sci. Res. Dev.*, Vol. 26 (2011) 247-256.

[11] A. Bhatele, L.V. Kale, S. Kumar. Dynamic Topology Aware Load Balancing Algorithms for Molecular Dynamics Applications-. In *Proceedings of the 2009 ACM International Conference on Supercomputing*, 2009.

[12] A. Bhatele, L.V. Kale. Heuristic-based Techniques for Mapping Irregular Communication Graphs to Mesh Topologies. In *Proceedings of Workshop on Extreme Scale Computing Application Enablement - Modeling and Tools*, 2011.

[13] T. Hoefler, M.Snir. Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *Proceedings of the 2011 ACM International Conference on Supercomputing*, 2011.

[14] A. Bhatele, L.V. Kale. Application-Specific Topology-Aware Mapping for Three Dimensional Topologies. In *Proceedings of IPDPS*, 2008.

[15] A. Bhatele, G. Gupta, L. V. Kale, I.-H. Chung. Automated Mapping of Regular Communication Graphs on Mesh Interconnects. In *Proceedings of International Conference on High Performance Computing (HiPC)*, 2010.

[16] K. Kandalla, H. Subramoni, A. Vishnu, D.K. Panda. Designing Topology-Aware Collective Communication Algorithms for Large Scale Infiniband Clusters: Case Studies with Scatter and Gather. *The 10th Workshop on Communication Architecture for Clusters (CAC 10), held in conjunction with International Parallel and Distributed Processing Symposium (IPDPS 2010)*.

[17] http://www.top500.org/

[18] E. Ma, L. Tao. Embedding among Toruses and Meshes. In *Proceedings of International Conference on Parallel Processing*, 1987, pp. 178-187

[19] E. Ma, L. Tao. Embedding among Meshes and Tori. *Journal of Parallel and Distributed Computing*, 1993, pp. 62-76

[20] S. Chittor, R. Enbody. Performance Evaluation of Mesh-connected Wormhole-routed Networks for Interprocessor Communication in Multicomputers. In *Proceedings of SC 90*, pp. 647-656.

[21] Lucas K. Wagner, Michal Bajdich, Lubos Mitas. QWalk: A Quantum Monte Carlo Program for Electronic Structure. *Journal of Computational Physics*, Volume 228, Issue 9, 20 May 2009, Pages 3390-3404, ISSN 0021-9991, 10.1016/j.jcp.2009.01.017.

[22] F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, F. Franchetti, S. Kral, C. W. Ueberhuber, and J. Lorenz. Large-scale electronic structure calculations of high-Z metals on the Blue Gene/L platform. In *Proceedings of Supercomputing*, 2006.

[23] Blake G. Fitch, Aleksandr Rayshubskiy, Maria Eleftheriou, T. J. Christopher Ward, Mark Giampapa, and Michael C. Pitman. Blue matter: Approaching the limits of concurrency for classical molecular dynamics. In *Proceedings of Supercomputing*, 2006.

[24] Abhinav Bhatele, Laxmikant V. Kale, and Sameer Kumar. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *International Conference on Supercomputing*, 2009.

[25] E. Zahavi. Fat Trees Routing and Node Ordering Providing Contention Free Traffic for MPI Global Collectives. *Journal of Parallel and Distributed Computing*, February 2012, ISSN 0743-7315, 10.1016/j.jpdc.2012.01.018.

[26] F. Pellegrini. Static mapping by dual recursive bipartitioning of process and architecture graphs. *Proceedings of SHPCC'94*, Knoxville, Tennessee, pages 486-493. IEEE Press, May 1994.

[27] George Karypis, Vipin Kumar. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering. *10th Intl. Parallel Processing Symposium*, pp. 314 - 319, 1996

[28] Shahid H. Bokhari. On the Mapping Problem. *IEEE Trans. Computers*, vol. 30, no. 3, pp. 207214, 1981.

[29] Pavan Balaji, Rinku Gupta, Abhinav Vishnu, Pete Beckman. Mapping Communication Layouts to Network Hardware Characteristics on Massive-scale Blue Gene Systems. *Comput. Sci. Res. Dev.*, (2011) 26: 247256, DOI: 10.1007/s00450-011-0168-y.

[30] Poonacha Kongetira, Kathirgamar Aingaran, Kunle Olukotun. Niagara: A 32-way Multithreaded SPARC Processor. *IEEE Micro*, 25(2):2129, 2005.DOI: 10.1109/MM.2005.35

[31] J. A. Kahl, M. N. Day, H. P. Hofstee, C.R. Johns, T.R. Maeurer, D. Shippy. Introduction to the Cell Multiprocessor. *IBM Journal of Research and Development*, 49(4):589604, 2005.

[32] Intel. From a few cores to many: A Tera-scale Computing Research Overview, 2006.

[33] Sailesh Kottapalli. Jeff Baxter. Nehalem-EX CPU Architecture. *HOT CHIPS 21*, 2009.

[34] C.E. Leiserson. Fat-trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, 34(10), October 1985.

[35] Natalie Enright Jerger, Li-Shiuan Peh. On-ChipNetworks. *Synthesis Lectures on Computer Architecture*, 2009

[36] Y. Ajima, T. Inoue, S. Hiramoto, T. Shimizu, Y. Takagi. The Tofu Interconnect. *IEEE Micro*, Volume: 32, Issue: 1, 2012

[37] T W Ainsworth, T M Pinkston. On Characterizing Performance of the Cell Broadband Engine Element Interconnect Bus. In *Proceedings of the International Network on Chip Symposium*, pages 1829, May 2007. DOI: 10.1109/NOCS.2007.34

[38] RCA: Managing System Software for Cray XE and Cray XT Systems - S239331

[39] H. Subramoni, S. Potluri, K. Kandalla, B. Barth, J. Vienne, J. Keasler, K. Tomko, K. Schulz, A. Moody, and D. K. Panda. 2012. Design of a scalable InfiniBand topology service to enable network-topology-aware placement of processes. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (SC '12). IEEE Computer Society Press, Los Alamitos, CA, USA, , Article 70 , 12 pages.

[40] P. Altevogt, H. Boettiger, T. Kiss, Z. Krnjajic. IBM Blade Center QS21 Hardware Performance. *TR, WP101245, IBM*, 2008.

[41] M. Kistler, M. Perrone, F. Petrini. Cell Multiprocessor Communication Network: Build for Speed. *IEEE Micro*, 26:10 23, 2006.

[42] T. Nagaraju, P.K. Baruah, A. Srinivasan. Optimizing Assignment of Threads to SPEs on the Cell BE Processor. *Technical Report TR-080906*, Computer Science, Florida State University, 2008.

[43] G. Okten, A. Srinivasan. Parallel quasi-Monte Carlo methods on a Heterogeneous Cluster. In *Proceedings of the Fourth International Conference on Monte Carlo and Quasi Monte Carlo (MCQMC)*. Springer-Verlag, 2000.

[44] M.K. Velamati, A. Kumar, N. Jayam, G. Senthilkumar, P.K. Baruah, S. Kapoor, R. Sharma, A. Srinivasan. Optimization of Collective Communication in Intra-Cell MPI. In *Proceedings of the 14th IEEE International Conference on High Performance Computing (HiPC)*, pages 488–499, 2007.

[45] Lester WA, Reynolds P, Hammond BL. Monte Carlo Methods in Abinitio Quantum Chemistry. *Singapore: World Scientific*, 1994. ISBN 9789810203214

[46] Foulkes WMC, Mitas L, Needs RJ, Rajagopal. G. Quantum Monte Carlo Simulations of Solids. *Reviews of Modern Physics*, 2001;73(1):33.

[47] Luchow A. Quantum Monte Carlo methods. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2011;1(3):388-402.

[48] Martin RM. Electronic Structure: Basic Theory and Practical Methods. *Cambridge University Press*, 2004. ISBN 0521782856.

[49] Needs RJ, Towler MD, Drummond ND, Rios PL. Continuum Variational and Diffusion Quantum Monte Carlo Calculations. *Journal of Physics: Condensed Matter*, 2010; 22: 023201.

[50] CHAMP. http://pages.physics.cornell.edu/cyrus/champ.html, 2011.

[51] QMCPACK. http://qmcpack.cmscc.org/, 2011.

[52] Nemec N. Diffusion Monte Carlo: Exponential Scaling of Computational Cost for Large Systems. *Physical Review B*, 2010; 81(3): 035119.

[53] Gillan MJ, Towler MD, Alfe D. Petascale Computing Open New Vistas for Quantum Monte Carlo. In *Psi-K Newsletter*, vol. 103. 2011: 32.

[54] Hu YF, Blake RJ, Emerson DR. An Optimal Migration Algorithm for Dynamic Load Balancing. *Concurrency: Practice and Experience*, 1998;10:467-483.

[55] Pothen A, Simon HD, Liou K. Partitioning Sparse Matrices with Eigen- Vectors of Graphs. *SIAM Journal on Matrix Analysis and Applications*, 1990;11:430-452.

[56] Barnard ST, Simon HD. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice and Experience*, 1994;6:101-107.

[57] Karypis G, Kumar V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 1998;20:359-92.

[58] Karypis G, Kumar V. Multilevel K-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 1998;48:96-129.

[59] Karypis G, Kumar V. A Parallel Algorithm for Multilevel Graph Parti-tioning and Sparse Matrix Ordering. *Journal of Parallel and Distributed Computing*, 1998;48:71-95.

[60] Karypis G, Kumar V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *Tech. Rep. 95-035; University of Minnesota*, 1995.

[61] Oliker L, Biswas R. PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes-. *Journal of Parallel and Distributed Computing* 1998;51:150-177.

[62] Cybenko G. Dynamic Load Balancing for Distributed Memory Multiproce-
ssors. *Journal of parallel and distributed computing*, 1989;7:279-301.

[63] Diekmann R, Frommer A, Monien B. Efficient Schemes for Nearest Neighbor Load Balancing. *Parallel Computing*, 1999;25(7):789-812.

[64] Hu YF, Blake RJ. An Improved Diffusion Algorithm for Dynamic Load Balancing. *Parallel Computing*, 1999;25:417-444.

[65] Ghosh B, Muthukrishnan S, Schultz MH. First and Second Order Diffusive Methods for Rapid, Coarse, Distributed Load Balancing. *Theory of Computing Systems* 1998;31:331-354.

[66] Elsasser R, Monien B, Schamberger S. Distributing Unit Size Workload Packages in Heterogeneous Networks. *Journal of Graph Algorithms and Applications*, 2006;10:51-68.

[67] Catalyurek U, Boman E, Devine K. A Repartitioning Hypergraph Model for Dynamic Load Balancing. *Journal of Parallel and Distributed Computing*, 2009;69:711-724.

[68] Walshaw C, Cross M. Dynamic Mesh Partitioning and Load-balancing for Parallel Computational Mechanics Codes. In *Topping BHV, ed. Computational Mechanics Using High Performance Computing*, Edinburgh: Saxe-Coburg Publications; 1999.

[69] Schloegel K, Karypis G, Kumar V. A Unified Algorithm for Load Balancing Adaptive Scientific Simulations. In *Proceedings of the IEEE/ACM SC 2000 Conference*, 2000.

[70] Iosup A, Sonmez O, Anoep S, Epema D. The Performance of Bags-of-tasks in Large-scale Distributed Systems. In *Proceedings of the HPDC*, 2008.

[71] Fujimoto N, Hagihara K. Near-optimal Dynamic Task Scheduling of Independent Coarse-grained Tasks onto a Computational Grid. In *Proceedings of the International Conference on Parallel Processing*, 2003:391-398.

[72] Maheswaran M, Ali S, Siegal H, Hensgen D, Freund R. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *Proceedings of Heterogeneous Computing Workshop*, 1999:30-44.

[73] Dhakal S, Hayat M, Pezoa J, Yang C, Bader D. Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration Theory Approach. *IEEE Transactions on Parallel and Distributed Systems*, 2007;18:485-497.

[74] R. A. Kronmal and A. V. Peterson. On the Alias Method for Generating Random Variables from a Discrete Distribution. *The American Statistician*, 33:214–218, 1979.

[75] A. J. Walker. An Efficient Method for Generating Discrete Random Variables with General Distributions. *ACM Transactions on Mathematical Software*, 3:253–256, 1977.

[76] J. L. Traff. Implementing the MPI Process Topology Mechanism. *ACM/IEEE conference on Supercomputing*, SC02, 2002.

[77] Y. Li, P.M. Pardalos, M.G.C. Resende. A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 16 (1994) 237-261.

[78] S. Arora, A. Frieze, and H. Kaplan. A New Rounding Procedure for the Assignment Problem with Applications to Dense Graph Arrangement Problems. *Mathematical Programming*, Vol. 92 (2002), 1-36.

[79] Pardalos, Panos M., Vavasis, Stephen A. Quadratic Programming with One Negative Eigenvalue is NP-hard. *Journal of Global Optimization, 1 (1): 1522, 1991.*